In compliance with the
Canadian Privacy Legislation
some supporting forms
may have been removed from
this dissertation.

While these forms may be included
in the document page count,
their removal does not represent
any loss of content from the dissertation.

AN INTELLIGENT E-MARKETPLACE FOR SOFTWARE COMPONENTS TRADING

By

Johnny Zhilie Yang

A thesis submitted to

The Faculty of Graduate Studies and Research

in partial fulfillment of

The requirements for the degree of

Master of Science

In Information and System Science

Ottawa-Carleton Institute for Computer Science

School of Computer Science

Carleton University

Ottawa, Ontario

May 22, 2003

© Copyright 2003, Johnny Z. Yang

National Library
of Canada

Bibliothèque nationale
du Canada

Acquisitions and
Bibliographic Services

Acquisitons et
services bibliographiques

395 Wellington Street
Ottawa ON K1A 0N4
Canada

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence

ISBN: 0-612-88790-1

Our file Notre référence

ISBN: 0-612-88790-1

The author has granted a non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of this thesis in microform, paper or electronic formats.

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de cette thèse sous la forme de microfiche/film, de reproduction sur papier ou sur format électronique.

The author retains ownership of the copyright in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

L'auteur conserve la propriété du droit d'auteur qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

Canada

The undersigned hereby recommend to
The Faculty of Graduate Studies and Research
Acceptance of the thesis,

AN INTELLIGENT E-MARKETPLACE FOR SOFTWARE
COMPONENTS TRADING

submitted by
Johnny Z. Yang

Dr. Frank Dehne
(Director, School of Computer Science)

Dr. Michael Weiss
(Thesis Supervisor)

Carleton University
May 22, 2003

Abstract

This thesis introduces a new e-marketplace for software components trading.

This e-marketplace is focused on providing decision support on whether buying/renting or when to sell software component for software developers. For buyers, a business decision model on whether to buy or rent a software component in current markets is studied and integrated into the e-marketplace as an automated decision-making function. For sellers, a decision problem on when to sell is examined and built into this e-marketplace. Current discounting interest rate, length of planned horizon, number of uses and maintenance cost of a software component are four key parameters of decisions making processes on whether to buy or rent the software component from the e-marketplaces.

Acknowledgements

I would like to express my sincere gratitude and appreciation to my supervisor, Dr. Michael Weiss for his guidance, inspiration and support during this research.

I would also like to thank my defense committee: Dr. Dwight Deugo, School of Computer Science, Carleton; Dr. Dorina C. Petriu, Department of System and Computer Engineering, Carleton University; for their time to read my thesis, and for their corrections and comments.

Last but not least, I would like to dedicate this thesis to my family for their encouragement, support during these studying years.

Contents

Abstract	iii
----------	-----

Acknowledgements	iv
------------------	----

1 Introduction	1
-----------------------	----------

1.1 Motivation	2
----------------------	---

1.2 Research goals and scope.....	4
-----------------------------------	---

1.3 Contributions	5
-------------------------	---

2. Background

2.1 Historical Reasons	7
------------------------------	---

2.2 Related Work	10
------------------------	----

2.2.1 Traditional licensing based software component marketplaces.....	10
--	----

2.2.2 Leasing based software component marketplaces	12
---	----

2.2.3 Why use mobile agent.....	16
---------------------------------	----

2.2.4 Current Agent Based e-marketplace Models	17
--	----

3 Software Component e-Marketplace Design 24

3.1 Design Assumptions of The e-marketplace	24
---	----

3.2 Business Decision Model of The e-marketplace	27
--	----

3.2.1 Trade off Between Buying and Renting Decisions on Software Components.....	28
---	----

3.2.2 Total Cost Decision Model for Buying or Renting Software Components	31
--	----

3.2.3 Selling Decision Model for Software Components	36
--	----

3.3 e-Marketplace Requirements and Architecture	36
---	----

4 Implementation of Software Component e-Marketplace 42

4.1 Brief Description of The Tryllian ADK	42
---	----

4.2 The Type of Agents and The Messages In The e-Marketplace	47
--	----

4.3 General trading Scenarios	48
-------------------------------------	----

4.4 Agents Functional Design	54
------------------------------------	----

4.4.1 User Agent	55
------------------------	----

4.4.1.1 Design and Implementation of User Agent	62
---	----

4.4.2 Confederation Agent	72
---------------------------------	----

4.4.3 Query Agent	77
-------------------------	----

4.4.4 Buyer Agent	81
4.4.4.1 Buyer Agent's Parameters	81
4.4.4.2 Detail Functional Design of Buyer Agent	83
4.4.5 Seller Agent	89
4.4.5.1 Functional Design of Seller Agent	90
4.4.6 Market Agent	96
4.5 Monetary, Security and Transaction System	103
5 Experimental Validation	105
Test Case 1. User Agent Marketplace Knowledge Validation	107
Test Case 2. Parameter Sensitivity Testing on Buying/Renting Decision of User Agent	108
Test Case 3. Selling Decision Test of The User Agent	115
6 Conclusion	118
5.1 Summary	118
5.2 Future Research	121
Bibliography	122

List of Figures

2.1 Web Page of WWW.COMPONENTSOURCE.COM	11
2.2 Web Page of WWW.SALCENTRAL.COM	14
3.1 Software components chain of production	25
3.2 Roles of traders in the marketplace	26
3.3 Total cost of Buying and Renting VS number of uses.....	29
3.4 Influence diagram for the buying/renting decision problem	32
3.5 Use Case Diagram of The e-marketplace	37
3.6 e-Marketplace general architecture	40
4.1 Functional Components Diagram of ADK	43
4.2 Sense-reason-act Loop Diagram of Agent	45
4.3 Message Sequence Diagram of the e-marketplace	50
4.4 GUI Interface of User Agent	55
4.5 GUI Picture of Decision-making to buy by User Agent	58
4.6 Buying Trading Result Picture of The User Agent	59
4.7 GUI Picture of Decision-making to rent by User Agent	60
4.8 Selling Trading Result Picture of The User Agent	61
4.9 UML Class Diagram of The User Agent	62

4.10 Tasks workflow Diagram of The User Agent	63
4.11 State diagram of the user agent	64
4.12 Sequence Diagram of User Agent's Tasks	65
4.13 Task Sequence Diagram of The Confederation Agent	75
4.14 UML class diagram of the Query Agent	78
4.15 Task Sequence Diagram of the Query Agent	79
4.16 Message Sequence Diagram of The Buyer Agent	84
4.17 State Diagram of The Buyer Agent	85
4.18 Message Sequence Diagram of The Seller Agent	91
4.19 GUI Picture of Trading Activity of The Market Agent	96
4.20 GUI Picture of Trading Statistic Result of The Market Agent	97
4.21 Task structure of the market Agent	98
4.22 UML Class Diagram of the Market Agent.....	98
4.23 Message Sequence Diagram of The Market Agent	101
5.1 Buying/Renting Decision VS Length of Horizon	110
5.2 Buying/Renting Decision VS Discounting Interest Rate	111
5.3 Buying/Renting Decision VS Number of Uses	113
5.4 Buying/Renting Decision VS Maintenance Cost	114

Chapter 1

Introduction

As software is becoming increasingly more object-oriented and modularized, more and more software applications are being built by assembling software components from various software vendors. This new trend in the software industry has inspired many software vendors to produce a variety of reusable software components. In order to foster this trend and promote the re-use of software components across the software industry, there is a need to establish an electronic marketplace (e-marketplace) for trading software components. We argue that an e-marketplace should not only be a trading place for software components, but should also assist the decision-making process of traders (software developers and vendors), allowing them to make the most suitable business decisions with respect to their respective needs.

Our main focus is on supporting the decision-making process of software developers. The e-marketplace helps them make a decision on whether to

buy or rent software component based on current market conditions and the vendor's risk preferences.

The implementation of the electronic marketplace follows a multi-agent approach. Each entity (buyer, seller, or market facilitator) is represented by an agent. This e-marketplace is designed as a federation of markets in which each market independently trades a number of software components, and traders can go to any number of markets to buy or sell software components.

This chapter presents the motivation for our work and the research goals of this thesis. It also includes with a list of the research contributions.

1.1 Motivation

Recently, the ASP (Application Service Provider) model and web service technology have opened up a new distribution model – software component leasing. In terms of the leasing, a software component is considered as a 'service' that can be provided to software developers on 'pay-per-use' basis over a network. In contrast, with the traditional software licensing model, a software component is an information merchandise that can be bought and used like any other goods on 'all-or-nothing' basis. Due to the emergence of the software leasing model, the software developers and the software component vendors now have a choice between licensing software

components and leasing 'services'. In [4], Marchand and Jacobsen have studied and developed an economic model that captures software selling versus leasing decision-making problem from the independent software vendor's point of view. The model underlined some complex interrelations between the traditional software market, based on licensing software applications, and the ASP-market, based on leasing software applications. Similarly, the software component buyers (software developers) also face a similar decision-making problem on whether to buy or rent a software component from marketplaces. Obviously, the decision on whether to buy or rent a software component will have a business impact on the user software development. This decision-making issue on the side of the user will be the main focus of our thesis. In this thesis, we use the term 'users' to represent software developers who would like to acquire (buying or renting) software components from marketplaces, and the term 'service providers' to represent software component vendors when they are leasing their software components in marketplaces.

On the other hand, as the demand for software components grows, there is a need to establish an effective e-marketplace to automatically trade the software components to software buyers and sellers, so that they can focus on their core business – developing software applications. By incorporating the decision-making on buying/renting into this e-marketplace, we can make this marketplace even more attractive to software developers.

Based on these considerations we define the following research questions:

- How can the electronic marketplace automate most of the trading processes for the software component buyers and sellers?
- How can this e-marketplace provide economic benefits to traders (software developers and vendors) based on their decision making process during the buying and selling of their software component activities?

1.2 Research goals and scope

The overall goal of this thesis is to propose and build a software component e-marketplace that will automate most of the market trading processes, and assist buyers and sellers in their buying/selling decision problems. For software developers (buyers), this e-marketplace will provide an automated decision-making capability (whether buying or renting a software component). For software vendors (sellers), this e-marketplace will provide an automated function on when to sell a software component into e-marketplaces.

Questions on how to allow secure payment for components trading, how to promote e-marketplaces, how to notify users about components of interest, or how to enhance security of the marketplace etc, are outside the scope of this research.

1.3 Contributions

We can point out the following issues as the contributions of this thesis.

- It integrates a decision-making model (whether to buy or rent a software component) into the e-marketplace, so that the buying/renting decision can be made in a fast and automated way based on marketplace's conditions. This model can automatically make a decision on buying /renting to benefit the software component buyers during market trading activities. This process can also free up software developers' time so that they can focus on their software development business instead of spending a lot of time to manually make decisions on buying/renting components in the marketplaces.
- It provides a sale price evaluation mechanism for the sellers. This mechanism can automatically protect the sellers from losing any money by selling a software component in marketplaces.
- It proposes a distributed e-marketplace structure for software components trading, so that software component buyers and sellers can easily select multiple marketplaces to trade.

- It automates most of the trading processes for the buyers and the sellers, so that the software component buyers and sellers do not need to manually browse, search, compare and select software components from different marketplaces.

This chapter described the motivation, goals and contributions of this thesis. The next chapter presents the background for this research and related work.

Chapter 2

Background

This chapter presents the background for this research and related work.

2.1 Historical Reasons

Back in the 1970's, most of the software was developed and used inside a specific company. The software designers in these companies developed every thing from ground-up, and software component re-use was limited to within the company. But as software became more and more sophisticated, this development model made the software development process very expensive. The solution is standardization and modularization of the software components. Standardization means making software components from different companies have a standard compatible interface so that software developers can easily and efficiently integrate them into their own applications. This standardization also makes software components from different companies comparable and tradable in marketplaces.

Ideally, at the end of this standardization, software application development will become a 'industrialized' process in which the various software components, sub-components and sub-sub-components of almost every final software product are built by specialists in a very long chain of production – which economists call an extended structure of product [1]. This software development process will much more resemble a car assembly line; software developers assemble various software components into a final application by calling standard component interfaces. This process will allow software vendors to simplify application development, speed-up the delivery of high-quality products to the market, and reduce the number of IT professionals required for development. This movement in the software industry is likely to grow even more as companies seek to produce highly competitive products while reducing costs.

However, in order for this standardization process to prevail, there are two major obstacles that we must overcome. The first one is incompatibility among software components from different companies. As we stated in the beginning, at the early stage of software development, every software company and individual had developed their own proprietary standards and platforms to build specific solutions for a particular set of problems. The incompatibility among these platforms has prevented software component re-use across different companies. But fortunately, as new technologies such as object oriented design, platform independent programming and web services

emerge, this incompatibility has gradually disappeared. More recently, as major software vendors incorporate compatible software component re-use technology into their new development platforms such as Microsoft®'s .NET, HP®'s eServices, Sun Microsystems®' Sun ONE strategy, the software chain of production by assembling sub-components has become a reality in many areas.

The second obstacle is the lack of effective marketplaces for reusable software components. If there are no effective markets for re-usable software components, then there are no effective means through which software specialists can share their experience and expertise by selling their products and no economic incentives to keep them producing more and more standardized reusable software components.

As stated earlier, as new technologies are working closely to overcome the incompatibility between various software components, we expect this incompatibility to disappear in the near future. Therefore the lack of effective software component marketplaces has become a key issue for industrialized software development process. Yes, there are web sites for buying and selling software components, but these sites require significant human involvement. One goal of an effective marketplace is to automate the interactions between buyers and sellers. Another goal is to support the decision making of the buyers and sellers, that is, help buyers make an

effective choice on which component to choose, and on whether to buy or rent this component, and help sellers price their components. In our research we have focused on building such an effective e-marketplace for software components.

2.2 Related work

The following is a review of software component marketplaces that are currently in operation on the Internet.

2.2.1 Traditional licensing based software component marketplaces

www.componentsource.com, www.internetcomponent.com and www.flashline.com are three representative software component marketplaces on the Internet. All of these marketplaces operate on a similar business model, so we will only study one of them - www.componentsource.com. Figure 2.1 is the index page of www.componentsource.com.

Figure 2.1 Index Page of WWW.COMPONENTSOURCE.COM

From Figure 2.1, we can see that, in marketplace www.componentsource.com, all software components are categorized based on their compatible platforms and standards such as COM, .NET, MTS, JAVA, J2EE, VCL, C++, etc. This marketplace is essentially an e-commerce web store that is operating on a similar business model as other online merchandise stores such as Amazon.com. Like all other web stores, it offers software component searching capability, secure payment channels, software component recommendation and evaluation services, etc. Most of the software components listed on this marketplace can be easily integrated into

the existing software applications on their corresponding platforms. Customers who want to buy these components will have to login into this marketplace web site, register themselves as users, manually search for the needed software components, compare their prices, and go through the payment process to purchase the components. As this marketplace is also open to Independent Software Vendors (ISVs) who produce software components, they also have to register themselves as software component sellers and fill out a set of forms to list their components in the marketplace. This marketplace will make profit by charging commissions to sellers for the components that they sold.

In conclusion, even though these web marketplaces provide a set of central places for people to buy or sell software components, the trading activities in these web sites are time-consuming and need great human intervention to browse, select, and evaluate the wanted software components. Moreover, these marketplaces only provide software component licensing option to buyers and sellers. Without the leasing option being available, buyers and sellers will have limited choice in their software business process.

2.2.2 Leasing based software component marketplaces

Web services technology based software component stores are appearing on the Internet recently as an attractive business model for leasing software

components to software developers. Web service technology is an emerging technology that is truly designed to work with multi-platform applications. Web service standards, such as SOAP (Simple Object Access Protocol), WSDL (Web Services Description Language) and UDDI (Universal Description, Discovery and Integration), have provided a solid platform for developing software applications in service oriented architectures (SOAs) [13]. This is a development paradigm and infrastructure in which software applications are broken into software components, which are then exposed as 'services' that can be dynamically discovered and invoked over the Internet. Web-services-based SOAs are expected to become more popular as increasing number of organizations have begun to use web services and deploy them more widely within their enterprises. As this happens, entire enterprise architectures will migrate from infrastructures based on monolithic applications integrated in point-to-point fashion, to those based on loosely coupled services distributed across the network. In SOAs, software components are 'services' that are used by software applications on per-use (rent) basis.

www.salcentral.com, <http://www.xmethods.com/> and <http://www.webservicewatch.com/> are representative marketplaces in which software vendors lease software components in web services format on pay-per-use basis. All of them are similar in terms of their business operations. So we only study one of them. Figure 1.2 illustrates the index page of www.salcentral.com

Figure 2.2 Index Page of WWW.SALCENTRAL.COM

From Figure 2.2, we can see that the marketplace www.salcentral.com acts as a centralized broker that helps 'users' (software developers) find their needed software components, and also provides 'services providers' (software component vendors) a place to advertise their software components. For users, this marketplace provides software component recommendations, service monitoring, searching capabilities and secure payment services. For service providers, the marketplace provides software component promotions, subscription and quality assurance services. A typical scenario is as follows. After a software component (a web service) is created in a software provider's site, the software provider will logon into this

marketplace to register this web service with the description of functions and the calling method. A potential user will either come to search for this web service or have been notified by the marketplace about this web service. After an evaluation of this web service, the user can rent this service by signing a contract that charges the user on pay-per-use basis. The user will be informed with a specific calling interface and a secure key in order to use this web service (software component) within their software applications. The service providers can also ask a third party to deploy and maintain this web service (software component) for them, in which case, the marketplace trading scenarios are also similar to what we previous described.

Generally speaking, these leasing based marketplaces are passive web sites that require buyers and sellers to spend a lot of time registering themselves, setting, searching and evaluating web services.

In conclusion, the previous two existing software component marketplace models (licensing model and leasing model) still require significant human efforts during trading activities. Moreover, even though they provide marketplaces for the developers to buy or rent software components, they do not provide support for the developers to make a decision on whether to buy or rent the software components. To provide this support, we propose a new software component e-marketplace, in which most trading activities will be automated and user decision-making support functions will be provided.

In our e-marketplace, there are a number of entities (buyers, sellers and marketplace managers etc.). These entities will have to interact with each other to accomplish trading activities such as buying or selling.

2.2.3 Why use mobile agents?

Software agents are autonomous computational entities (autonomy), which interact with their environment (reactivity) and other agents (social ability) in order to achieve their own goals (proactive) [12]. As a special case of software agents, the characteristics of mobile agents in mobility and low cost of transmission within networks have made them widely used in constructing e-marketplaces. In an e-marketplace network, if negotiation interactions among buyers and sellers are carried out over network communication channels, then these interactions can generate huge network traffic. Mobile agents allow these interactions to be packaged together and sent as a discrete piece of network traffic. This then allows all the interactions to take place locally. Also due to mobility of the mobile agent, it is particularly suitable to represent an entity (buyer or seller) which needs to travel through multiple places in a distributed structure.

On the other hand, in our e-marketplace, we need to represent a number of independent entities. Mobile agents are autonomous and are able to encapsulate all the required data with them, and hence do not need to

communicate with other computers. Moreover, though mobile agents are very small in size, they can grow as they need to accommodate more data. This is perfectly suited for negotiations in marketplaces. These characteristics of mobile agents have made them ideal candidates for representing an independent entity in marketplace.

Because of these advantages of mobile agents in e-marketplace construction, our e-marketplace will also be implemented using mobile agent technology.

2.2.4 Current Agent Based e-marketplace Models

Since this e-marketplace will be implemented using mobile agent technology, it is important for us to review current representative marketplace models that are built on various software agent systems. The following section briefly illustrates these marketplaces' architectures and methodologies.

Kasbah [9] is an agent based virtual marketplace that has been developed at the MIT media lab. Essentially, Kasbah is a web site in which people can sell or buy goods by creating selling agents or buying agents that interact with each other in the marketplace to get a best deal for their users. Users can create a buying agent or a selling agent by specifying certain parameters that guide and constrain the agent's overall behaviors. These parameters include the desired price, the lowest or the highest acceptable price, time allowed to

trade, price adjusting functions etc. Kasbah's marketplace matches up agents interested in buying or selling things of the same kinds. When a selling agent is created, the marketplace asks what the agent is interested in selling. The marketplace then sends the agent a list of all potential buyers for that particular item. The marketplace also sends messages to all of these potential buyers informing them of the existence of the new selling agent. The same thing happens when a buying agent is created. When an agent leaves the marketplace, the marketplace notifies all of its potential customers (if it is a selling agent) or all the agents for whom it was a potential customer (if it is a buying agent). When the buyer agent and seller agent reach a deal, they ask their human owner to approve before the final deal can be made.

Kasbah greatly frees up the human's time from having to finding, negotiating and dealing with buyer or seller, it also learns the comprehensive list of potential buyers and sellers in the market place and achieves better price deal for its users.

AuctionBot [10] is a well-known experimental Internet auction market site developed at the University of Michigan. It provides both interfaces to human and software agents for trading in the market. Software agents in AuctionBot can be created using an agent API with a set of specified auction parameters. The created software agent communicates with the auction database. After the agent places a valid bid, a scheduler program (which is a daemon

process that continually monitors the database for auctions that have events to process or bids to verify) forks the appropriate auctioneer program. The auctioneer loads the auction parameters and the set of current bids from the database. The auctioneer validates bids as necessary, and may do one clear and/or one price quote each time it runs. After the auction finished, the result is informed to the agent.

AuctionBot is basically an Internet-based auction server. It supports a wide range of auction types, and it is also able to manage a large number of simultaneous auctions.

InterMarket[14], developed at Fredrich Schiller University aims at enabling mobile access and automated trading in e-marketplaces based on integration of mobile agents and intelligent decision-making stationary agents offered as an add-on component to a commercial e-marketplace platform. InterMarket uses two kinds of agents. One agent is a stationary intelligent trading agent that automates the users' decision-making and negotiation tasks in e-marketplaces such as searching, matching, comparison, bidding, negotiation and ordering with other agents and users. Another agent is a mobile agent that provides mobile access and communication to E-marketplaces from mobile devices such as Personal Digital Assistants (PDA) or Mobile Phones and also delivers initial trading instructions to stationary agents in the marketplace and reports the trading result to the user. InterMarket's trading

agents simulate human traders' behavior provided by an automated reasoning mechanism using human traders' knowledge and instructions that allow the trading agents to get involved in different trading tasks in the e-marketplace in a similar fashion as the human participants could.

InterMarket is a market system that integrates mobile agents and stationary agents with decision-making capability. This decision-making ability is built into the trade agent for trading decisions in the market activities. Mobile agents in this system are responsible for delivering messages between user and trading agents, which can reside on various mobile devices (PDA, Mobile phone etc.).

MAgNET (Mobile Agents for Networked Electronic Trading)[15] is a mobile agent based system prototype developed at University of California with Java and IBM Aglets SDK to enable buyers to comparison shop for items from different online sellers.

In MAgNET, a buyer creates one or more mobile shopping agents whose objective is to find a set of suppliers who are able to provide sub-components that are required for manufacturing the product and that are compatible with the buyer's production schedules. The shopping mobile agent asks a quota from every supplier it visited. If a quotation is attractive, the mobile agent may reserve the component before moving to the next site on its itinerary. Then

the mobile agents come back to the buyer with the quotations and reservations they have made. The buyer determines the best combination of the offers, and sends messages or agents to the suppliers to commit or cancel the reservations. This algorithm may cause a ripple effect through the entire supply chain.

On the other hand, with the similar mechanism, a supplier can create and dispatch a mobile agent to visit buyers with an offered price for the product, and the current quantity available for sale. Depending on the price offered, the buyer at that site determines whether to purchase and how much to purchase, based on the buyer's demand for the product. When the entire inventory is exhausted, or all of the buyer sites have been visited, the agent returns to the supplier site from which it had originated, with residual stock if any.

BiddingBot [17] is a multi-agent system developed at Nagoya Institute of Technology, which can support attending, monitoring, and bidding in multiple auction sites. It consists of several cooperative bidding agents that have been implemented with a mobile Java-based agent framework called MiLog.

In BiddingBot, multiple bidding agents can attend different auctions and bid on behalf of users simultaneously in order to obtain the items at the best price. BiddingBot's bidding agents can bid according to autonomous and

coordinated bidding mechanisms designed for the agents. The bidding support process consists of the following steps: The user inputs keywords describing the desired item to the system. The bidder agents gather the reasonable market price on the item by gathering information based on the keywords. The users decide a valuation of the item based on the item's approximate market price. The bidder agents cooperatively bid to the several auction sites facilitated by the leader agents, and the leader agents also inform the user whether the item can be awarded or not.

Each of the above discussed agent based e-marketplaces has its own strengths and weaknesses, but since every marketplace is designed to address a particular set of the problems, none of their structures can be directly applied to our software component e-marketplace. Kasbah is designed specifically for people to search for classified Ads. AuctionBot is basically an auction server in which agents are used to pass the bidding instructions from users to auction programs. Its centralized design did not address scalability issue of the system. InterMarket is a system that employs mobile agents and intelligent stationary agents together to deliver automated trading in marketplace, but only its stationary agents make trading decision in the marketplaces. Its mobile agents have no ability to collect market information to make business decisions and only act as a message passer between human user and stationary trading agent in the market. MAgNET system employs a salesman visiting-like structure in which either a buyer

agent or a seller agent visits a number of sellers or buyers one by one to buy or sell products. BiddingBot system sends out multiple mobile agents bidding in multiple sites to get the best price for the human user but it does not provide an automated decision support for the user. Because our research interest and goals are different from all of the above agent-based marketplaces, we are proposing a new marketplace structure that is scalable and integrated with decision capabilities (whether to buy or rent etc).

This chapter described the background of our research project, reviewed current software component marketplace and agent-based e-marketplaces. The next chapter introduces structure of our software component e-marketplace and a business decision model that is integrated into our decision-making functionality of the e-marketplace.

Chapter 3

Software Component e-Marketplace Design

In this chapter, structure of our software component e-marketplace will be described. Design assumptions of this e-marketplace and a business decision model will also be described in this chapter.

3.1 Design Assumptions of The e-marketplace

Based on current software technology trend, we can broadly predict that the future sophisticated software applications will be largely developed based on assembling different software components. Naturally, as competition grows in software components business, there will be a large number of software components with similar functionality produced by different vendors, which will have different performance and security features etc. Evaluation of these similar software components will be varied from users to users due to their different requirements for performance, security and maintainability. To

simplify this evaluation issue in our e-marketplace, we will make an assumption that every software component that has similar functionality is identical type of software component. As long as the software components provide the same functionality, we will consider they are the same products. Only differences between them are their producer and prices.

In this e-marketplace, we assume that profitability will be primary driving force for all software vendors and software developers to trade software components in this marketplace. Based on the chain of software production model described in section 2.1, we have drawn Figure 3.1 to illustrate this software components assembling process.

Figure 3.1 software components chain of production

From Figure 3.1, during a software application development process, any software components vendor in this chain can be buyer or supplier to other vendor based on its own needs. In other words, if one software vendor need

to buy sub-components from other vendors, these sub-components' vendors in turn will ask for sub-sub-components from another vendors. This ripple effect will form this chain of software production process. Therefore, each software vendor or software developer can be seller or buyer in this e-market at any point of time depending on their business position. We will call them all traders, and they can be classified in three roles - end sellers, middle buyer/seller and end buyers. These roles are illustrated in Figure 2.2.

Figure 3.2 Roles of traders in the e-marketplace

From Figure 3.2, obviously, every participant in this chain of the software production would like to make profit in their software component trading business. Based on our previous assumption that those similar software

components from different vendors are identical goods, so each buyer's primary consideration will be the price of the software component. In other words, buyer's decision on whether to buy or rent will be based on the cost of the software component in our marketplace. For software component sellers, of course, the highest selling price will be the ultimate goal.

Since this e-marketplace is designed for buying and selling software components in a chain of software production, the granularity of all software components will be varied from small to big. We assume in this marketplace, the sellers will compete with each other on same granularity of components. And buyers will get exactly what functionality they want. Based on above assumptions, we will use a continuous double auction to model this e-marketplace, in which the price of the software components is the only decisive criteria on matching buyers and sellers.

As we stated in our research goals, our intention of this e-marketplace project is to build an e-marketplace that will provide the automated business decision-making processes to benefit buyers and sellers. We will first illustrate the underlying business-decision model in next section, then introduce our e-marketplace overall structure.

3.2 Business Decision Model of The e-marketplace

As we described in the previous section, for every activity the buyers and sellers undertake in this software component e-marketplace, their final goal is to maximize their profit. For buyers, choosing whether to buy or rent software component under current market conditions is their goal to cut their development cost. For sellers, the decision on when to sell can prevent them to lose money in the e-marketplaces.

3.2.1 Trade off Between Buying and Renting Software Components

The traditional software license business model (buying) has been practiced for many years. But the concept of renting software has only appeared a few years ago as ASP (Application Service Provider), web service and new channel technology for software distribution (such as NetActive technology - www.netactive.com) emerged. Since then, software renting (pay-per-use basis) has become an important alternative choice for software buyers. The decision on whether to buy or rent a software component will have to base on buyer's business plan and current market conditions. In order to understand the benefit of buying and renting software components, we will study their impact and relations using a total cost function model.

If we refer TC as the total cost of renting and buying options, t as time variable, the interest rate as r .

For buying: $TC(r, t) = \text{initialBuyCost} + \sum \text{MaintenanceCost}(r, t)$

For renting: $TC(r, t) = \sum (\text{RentalCostEachTime} * \text{numberOfUses})(r, t)$

Interest r is one of key factors to affect the total cost of buying and renting, we will discuss its role in more detail later. For now, we will assume it is steady over the time t . Figure 2.3 below illustrates the total cost for buying and renting software component as function of the number of uses. In the diagram we assume the interest rate r and initial cost for buying and renting steady over the number of uses.

Figure 3.3 Total cost of Buying and Renting VS number of uses

From Figure 3.3, we can see that an initial cost of renting is low, but the total cost of renting will increase rapidly along the number of uses, whereas the

overall cost of buying will grow much slowly along the number of uses. Eventually the renting cost will pass the buying cost as the number of uses reach a certain value, that point is break-even point for this function. The break-even point is where that the total cost of renting is equal to the total cost of buying. Depending on buyer's needs and market conditions, buying and renting software components both have advantages and disadvantages to the buyer.

General speaking, renting has following advantages over buying.

1. *To conserve cash.* Rather than paying the entire cost of a software component up front, a renting requires a relatively small payment.
2. *To not be locked into old technology.* In software component business, every thing changes rapidly. When you rent a software component, you pay only for the every time you use it; in many cases, it is easy to upgrade to a new version at little or no additional cost. You don't have to be bothered with selling the old component when you're done with it.
3. *Budget saving.* Software component renting include maintenance and other services in a single price. You don't need to hire additional IT staff to install and maintain it [6].

On the other hand, buying option has some advantages over renting too:

1. *Exclusive usage.* In case you use a software component a lot in your software application, the buying option will be an economical choice. Because in software component business, renting is charged on per use basis.
2. *Security concern.* If a software component is needed to process some sensitive business data, the buying option is great way to protect these data.
3. *Performance.* Buying a software component and integrating it into your own software application clearly has higher performance over the renting due to usual network latency.

Because buyers have different business plans, determining whether renting or buying a software component can be a complex and time-consuming process. Following, we will introduce our business decision model that has been integrated into our e-marketplace.

3.2.2 Total Cost Decision Model for Buying or Renting Software Components

Making decision on renting or buying a software component can be a difficult task in a constantly changing marketplace. Our decision model is based on the total cost calculation for buying and renting a software component, therefore we call it 'total cost decision model'. There are a number of

parameters to be considered in this calculation such as buyer's *planning horizon*, current *discounting rate*, *number of uses*, *maintenance cost*, *rent cost per use* and *purchase cost*. Because this e-marketplace is a software simulation, so to simplify our discussion and calculation, we will use an abstract term 'time-unit' to represent time instead of using days, months and years. Following Figure 2.4 depicts a diagram for the buy/rent decision-making problem.

Figure 3.4 diagram for the buying/renting decision problem

From Figure 3.4, we can see that rental cost, discount rate, maintenance cost, purchase cost, number of uses and length of planning horizon each

directly influence total cost [2]. Rent/buy decision will also influence the total cost of buyers.

The above parameters are explained below:

- *Rental cost per use* – this parameter is a cost for software buyers each time a software component is rented on a ‘pay-per-use’ basis.
- *Length of planning horizon*. This parameter is based on buyers’ business plan on how long a software component is planned to use in the buyers’ software application.
- *Discounting interest rate*. This parameter refers to the depreciation rate for software component and it also refers to the value that could be obtained by not making an investment in software component.
- *Number of uses*. This parameter indicates the planned number of the usage for a software component within each time-unit of planning horizon.
- *Maintenance cost*. This parameter is the cost of maintaining a software component within each time-unit of planning horizon if buyers plan to buy this software component.
- *Purchase cost*. This is one time cost for buyer to purchase the software component.

Based on these parameters, our decision algorithm in this model is as following:

First of all, we calculate the total present value for renting option during length of the planning horizon. Rental cost per planning horizon time unit will be rental cost per use times the total number of uses per time unit,

$$\text{rentCostPerTimeUnit} = \text{RentCostPerUse} * \text{numberOfUses},$$

With a number of the time units in planning horizon and interest rate, we convert each of rentCost to present value. In other words, we are converting future cost of renting to present cost of the buyer for each time period in length of planning horizon. If we refer to the discounting interest rate as r , to the length of planning horizon as p , L as actual value of p , we have following result:

$$\text{TotalPresentCostForRent} = \sum_{p=1 \text{ to } L} ((\text{rentCostPerTimeUnit}) / (1 + r)^p)$$

This calculation will tell us how much the total future renting cost is worth in today's money.

Secondly, we calculate the total present cost of buying. For buying a software component option, besides paying a one time purchase fee, we will have to pay maintenance cost for this software component within every time-unit of length of horizon. Similar to renting cost, the total future maintenance cost can be converted to present value by following calculation:

$$\text{PresentCostForMaintenance} = \sum_{p=1 \text{ to } L} ((\text{maintenanceCostPerTimeUnit}) / (1 + r)^p)$$

Then total present value for buying option:

$$\text{TotalPresentCostForBuy} = \text{purchaseCost} + \text{PresentCostForMaintenance}$$

Thus we have two important present values: **TotalPresentCostForRent** and **TotalPresentCostForBuy**. By comparing these two values, we can make our decision based on whichever option has less cost. In case they are equal, we will choose rent option so that we can conserve some case in the front.

Software component renting shifts the burden of maintaining, upgrading and managing tasks to 'service provides'. But Software developers who bought a software component will have to maintain and manage themselves. In this model, if a software component is bought, we assume there is no salvage value for this software component at the end of the planning horizon. The developers will have to pay a full price to buy a new version of the software component to replace the old one. This replacement cost is represented as a new purchase price for this new version of the component.

This model is specifically designed for software component marketplaces. In this model, the software component's maintenance cost is steady over the time, which has nothing to do with the number of uses, that is different from normal commodity. Also there is no salvage value for the component, this is another difference between software component and normal goods.

3.2.3 Selling Decision Model for Software Components

In software component chain of production, software component sellers can buy some sub-components to assemble them to produce advanced components, and then sell them back to the marketplaces to make profit. Our selling decision on when to sell will be based on the following calculation:

If we refer the number of sub-components as n , and its value as N ,

$$TotalsubcomponentsCost = \sum_{n=1 \text{ to } N} (CostOfEachSubComponent_n)$$

Once we know the total cost of all the sub-components used to build current selling component, we can compare this *TotalsubcomponentsCost* and current market average selling price of the selling software component. If the *TotalsubcomponentsCost* is bigger than current selling price, which means the software vendor most likely won't make any profit at this component at present time, the decision will be made to stop selling action to prevent the seller losing money.

3.3 e-Marketplace Requirements and Architecture

As we stated in chapter 1, our research goal is to build an open software component e-marketplace that will shorten buyers and sellers market learning process and assist them on complicated buying/selling decision problems.

Base on this objective, the following Figure 3.5 of Use Case Diagram illustrates the e-marketplace requirements.

Figure 3.5 Use Case Diagram of The e-marketplace

From Figure 3.5, we can see the e-marketplace will provide following basic functions.

- Discovering e-marketplaces. Because software component traders usually locate at different networks in the Internet, this function is important for traders to discover proper marketplaces in order to trade their software components.
- Decision making for buying/renting. This function will provide intelligent decision-making capability to buyers who want to acquire a software component from the e-marketplaces. Its decision model has been illustrated in previous section 3.2.2.
- Decision making for sell. This function is to provide decision-making to sellers who want to sell a software component into marketplaces. Its model is illustrated in section 3.2.3.
- Buying/Renting software components trading activities.
- Selling software components trading activities.
- Reporting trading result function to traders.
- Query market condition function is used by Decision making functions to get current average market prices for software components.
- Matching sellers & buyers function.
- Demonstrating market activities.

From the above functions of the e-marketplace and our research goals, we derive following requirements for our e-marketplace.

1. This marketplace must be accessible from different networks in the Internet. Because software component traders are widely located in

different networks, a distributed market structure will fit this requirement, in Which many e-marketplaces distributed located in different networks, every one of them will operate independently.

2. Automated intelligent decision making functionality must be provided to software component traders, so that they can be freed from complex business decision processes in the constantly changing e-marketplaces.
3. Each e-marketplace must be able to trade a number of different software components, and one type of software component must be able to trade in multiple e-marketplaces.
4. Traders can freely go to any number of e-marketplaces to buy or sell a software component and specify their desired price ranges for a trading software component. During trading processes, buyers and sellers are able to adjust their trading prices within this range according to market conditions.
5. Locations of the e-marketplaces can be automatically discovered for traders.
6. The e-marketplace must have ability to demonstrate trading activities and statistic data on traded software components.
7. A trading result must be reported to human user for each traded software component.

Based on these requirements, we can design our market overall structure as shown in Figure 3.6.

Figure 3.6. e-Marketplace general architecture.

From Figure 3.6, we can see there are a number of e-marketplaces located at different sub-networks such as network 1, network 2 ... network n. For each e-marketplace, it can trade many types of software components, and one type of software component can be traded in multiple e-marketplaces. Thus, every trader who wants to buy or sell one a software component could visit multiple e-marketplaces.

In this e-marketplace, we have identified a number of independent entities that needs to interact with each other. First of all, there must be a marketplace manager entity that is able to match buyers and sellers, to advertise types of trading software components, and to demonstrate trading activities. Secondly, buyers and sellers in this e-marketplace have to be defined as different entities because they will have different tasks in e-marketplaces, and they must have ability to move from one e-marketplace to another. Thirdly, automated intelligent decision-making capabilities should be carried out by a separate entity that can interact with buyers/sellers, and able to interface with human users. At last, an independent entity is needed to discover locations of distributed e-marketplaces for software component traders. All these entities will be implemented using software agents in this e-marketplace project.

This chapter described the design assumptions of our e-marketplace, and its overall structures. The business decision model for buying or renting software component is also described. The next chapter will present the detail implementation of this marketplace that is based on a mobile agent system.

Chapter 4

Implementation of Software Component e-Marketplace

Based on our e-marketplace architecture and design requirements, we have implemented this e-marketplace with the Tryllian Agent Development Kit [5]. This agent development kit (ADK) is based on mobile agents. This chapter will give a description of the Tryllian ADK, and detail implementation of this e-marketplace.

4.1 Brief Description of The Tryllian ADK

Tryllian agent development kit (ADK) is developed based on the Java technology. The ADK consists two parts, the Agent Foundation Class (AFC) and the Agent Runtime Environment (ARE). Following Figure 3.1 illustrates functional components of ADK.

Figure 4.1 Functional Components Diagram of ADK.

The Agent Foundation Classes (AFC) is the main programming API and thus contains most of the interfaces and classes needed to interact with the agent-based elements. Using the AFC, you can create your own agents, create tasks for your agents to perform, create messages for your agent to send to other agents and create your own agent languages, and log your agent's activities.

The ARE is the kernel of the ADK. As an ADK user, interaction with the ARE takes place indirectly. The ARE presents itself to you through its System Agents, with which your agent can communicate. These system agents never leave the room (a container that holds agents and their resources) they are in

and provide a number of basic services such as moving agents to another room, keeping track of the agents in the room, etc. All communication between individual agents and the system is implemented by the ARE in the form of messages sent to a number of system agents. The ARE also implements the ADK's security features.

Every agent in Tryllian platform is living in a habitat (an application on which agents, rooms, and messaging, etc. run). A habitat is a collection of rooms that share a common code base and a Java Virtual Machine (VM). An agent has the ability to move itself to another location, but the location needs to have a habitat installed [5]. Tryllian agents do not have the ability to travel to sites that do not have a habitat installed.

As Tryllian agents interact with each other, the design of their proper behaviors during interactions will be major factors to achieve the overall system design goals. Tryllian agent's behavior is implemented using tasks. The composition of different tasks leads to the desired behavior of the agent. Conceptually, Agent behavior is governed by the sense-reason-act loop as shown in Figure 4.2.

Figure 4.2 Sense-reason-act Loop Diagram of Agent

In Figure 4.2, Sensing corresponds to receiving either a message or a heartbeat (that is, a periodic notification event to agent); reasoning corresponds to executing a certain task; and acting corresponds to either moving or sending a message. Generally speaking, the agent keeps track of which tasks are active and interested in certain events. A task defines what to do with an event, depending on the state of the task. A task may update the agent's internal state, or execute an action like sending a message. After the action has been scheduled, the task remains dormant until the next event arrives [5]. A task may finish. It is then removed from the agent's behavior. A task's internal states include Just Created, Active, Going to succeed, Going to fail, Success and Fail. Execution of tasks is done either in parallel or serially (scheduled tasks). When a certain sequence of actions is required, tasks can be organized in a scheduler (task scheduler).

The *TaskScheduler* is a special type of task that can be used to organize tasks in a logical manner. A TaskScheduler contains a number of subtasks. Of these subtasks, only one is active at any one time. The TaskScheduler allows you to specify the order in which tasks are executed and supports any control flow that can be specified using a state diagram. When you add a task to the task scheduler, you can specify explicitly which task should be invoked when this task succeeds, and which one should be invoked when it fails. Alternatively, you can provide a scheduling rule, which allows you to write your own code to retrieve the next task for success and for failure. The decision to retrieve a new task is usually based on the state of the previous task (failed or succeeded) [5]. Since a task scheduler itself is a task, you can nest task schedulers to encapsulate complex behavior of the agents. We have used a lot of task schedulers to achieve a number of goals in our software component e-marketplace.

It is very important for agents speak the same language during their communication. Tryllian agents follow FIPA protocol to build their messages. The content part of a Tryllian message is as following:

```
performative
subject
arguments ( optional )
```

Perfromative in Tryllian agent message is a subset of FIPA message protocol, which is a set of rules that agents use to communicate with each other. It includes: *request, query-ref, subscribe, cancel, inform, agree, refuse, failure and not-understood* etc.

Subject of the Tryllian message can be defined by developers, which basically means what topic this message is interested.

Arguments are usually content of the message. For example, when an agent receives a message with the current time, the performative of that message will be *inform*, the subject will be *time* and the arguments are the *hour*, *minutes* and *seconds* of the current time.

4.2 The Type of Agents and The Messages In The e-Marketplace

Using the Tryllian ADK, we have developed six different types of mobile agents in our e-marketplace:

User agent is a stationary intelligent agent that will be responsible for making business decisions for human users. And it will also provides a GUI interface for human users to input necessary parameters and displays trading results. User agent never moves out of its room.

Buyer agent is a mobile agent created by the User agent. It represents the User agent to perform buying or renting software component in the e-marketplaces. It will move around the e-marketplaces to buy or rent a software component. Once it finished its task, the buyer agent will move back to report the trading result to the User agent.

Seller agent is a mobile agent that is also created by the User agent. It represents the User agent to sell and lease software components in the e-marketplaces. Similar to the buyer agent, it will move from one marketplace to another to sell and lease a software component. Once its trading time is up, the seller agent will come back to the User agent to report the trading result.

Market agent is a stationary intelligent agent that is managing an e-marketplace in which a number of software components are traded. It will use a double auction algorithm to match buyers and sellers. It provides a GUI to demonstrate market activities.

Query agent is mobile agent that is created by the User agent. Its duty is to find out the current market average prices (selling price and renting price) for a specific software component.

Confederation agent is a mobile agent that is created by the User agent, whose task is to discover the addresses of the software component e-marketplaces for the user agent across the networks. Its algorithm will be discussed in the confederation agent section.

During their market activities, the above six agents will have to interact with each other to achieve their tasks. Their message format will follow the FIPA

message format that we have described in section 4.1. Based on our own e-marketplace's requirements, We have designed four subjects of the messages for these agents:

BUY_SUBJECT – this subject is used when buyer agents talk with market agents and user agents during buying/renting activities.

SELL_SUBJECT – this subject is used by seller agents to talk with market agents and user agents during their selling missions.

INFO_SUBJECT – this subject is used by confederation agents when they talk to each market agent in networks to discover the marketplaces' locations.

AVEPRICE_SUBJECT – this subject is used by query agents to talk to market agents to query market average prices of software components.

Following section will illustrate detail trading scenarios and interaction among these six agents in this e-marketplace.

4.3 General trading Scenarios

As we described in previous section, the e-marketplace will have 6 types of software agents interacting with each other during trading processes. Following Figure 4.3 illustrates a message sequence diagram in this e-marketplace.

Figure 4.3 Message Sequence Diagram of the e-marketplace

From Figure 4.3, a general work flow of the e-marketplace scenarios can be described as following:

At beginning, a number of **market agents** (marketplaces) will be created in different rooms on the same or different computers across networks. They will be initialized with what software components will be traded in their marketplaces. These market agents will provide GUI interfaces to demonstrate market's trading activities. After we have marketplaces established in the network, a number of user agents will be started. The user agents will also provide GUI interfaces to the human users, in which a number of parameters can be entered by the human users. These parameters include name of the software component, current discount interest rate, length of planning horizon, number of uses, estimated maintenance cost, prefer trading price, lowest or highest acceptable price, time window of trading, sub-components used in build this software component. The values of these parameters will be used in user agents' decision-making functions.

When a user agent is created, a number of **confederation agents** will be automatically created by the user agent. Then these confederation agents will be sent out to the different networks to acquire information on locations of marketplaces (market agents) and on what software components these marketplaces are able to trade. After these confederation agents reported the information, the user agent will be able to build a view of the marketplaces in the networks. The confederation agents then will be automatically destroyed.

At this point, the **User agent** will wait for the human user's instructions. If the human user would like to buy or sell a software component in the marketplaces, he or she will need to input some values for a set of parameters from the user agent's interface. After he or she issues command on whether to buy or sell a software component, a number of **Query agents** will be automatically created, these query agents will go to the marketplaces to query current average prices for selling and renting the software component. After these query agents reported the market prices to the user agent, they will be destroyed to save memory space.

If a buying request is issued from the human user, and the user agent has decided on whether to buy or rent the software component, a mobile **buyer agent** will be automatically created by the user agent. This buyer agent will be equipped with information on corresponding marketplaces, preferred marketplaces' locations, preferred buying or renting prices, and the highest acceptable price and time window (maximum time period for this trading activity). The buyer agent then will move around a number of marketplaces to buy or rent the software component in a desired price.

When the buyer agent arrives at a marketplace, it will send a buying or renting request to **market agent**. The market agent will then try to match a seller based on buyer's offered price. If a match is found, the market agent will inform the buyer agent with the seller's information such as seller's

companyID. At meantime, this seller will be put into a sold list by the market agent. If the market agent couldn't find any match for the buyer, the buyer agent will also be informed as not found, the market agent will also tell the buyer agent current market average price for this software component. The buyer agent will move to the next marketplace to do buying or renting again until a match is found or time is up. When the buyer agent find the software component or a valid trading time (time window) is expired, the buyer agent will move back to its originated room to report its trading result to its creator – the user agent. Then the user agent will demonstrate the trading result to the human user.

Similarly, for selling request from the human user, the user agent will make when to sell decision on a software component. If there is a profit on selling this software component in current marketplaces, a **seller agent** will be created with a desired selling price, the lowest acceptable price, locations of marketplaces, time window and seller's companyID. Otherwise, no seller agent will be created, selling request will be rejected. Once the seller agent is created, it will move to marketplaces to sell and lease the software component. The seller agent will send selling messages to a number of market agents with its selling and leasing prices. The market agents will reply with information on whether the component has been sold or not and current market average selling price. After its time window is expired, the seller agent will come back to its originate location to report selling result to the user

agent. The user agent will then display this trading result to the human users. The buyer agents and seller agents will be destroyed after they reported the trading result to the user agent.

In order to simplify our market monetary transaction process, this e-marketplace uses virtual money as a medium for buyers and sellers to exchange their software components. Because this marketplace is the prototype simulation software, we will initialize a certain amount of virtual money for each user agent when they are created. Thereafter, the user agents will have to manage this money to buy and sell software component. The market agents will not be given any virtual money, they will make money by charging commissions to sellers. Also to simplify our discussion, in this e-marketplace project, time will be measured in an abstract time unit form instead of using days, months and years etc. Next section we will examine detail functional implementation of these six types of agents.

4.4 Agents Functional Design

In this section, a detail design and implementation of six types of agents (User agent, Buyer agent, Seller agent, Market agent, Confederation agent and Query agent) will be discussed and presented.

4.4.1 User Agent

User agent is an agent that provides a GUI interface and makes business decisions (whether to buy/rent or when to sell) for the human users. The user agent's GUI interface is shown in Figure 4.4 below.

Figure 4.4 GUI Interface of User Agent

From Figure 4.5, we can see there are a number of parameters that are requiring the human user to input.

- *Software Component.* This parameter is a name of a software component that the human user would like to trade (buy/rent or sell) in marketplaces.

- *Length of planning horizon.* This parameter specifies the duration of a particular decision problem (such as buying or renting decision problem). In other words, this value indicates how long the human users plan to use this software component in their software applications.
- *Discounting interest rate.* This discount interest rate is a time value that the human user can make by not investing in the software component. In other words, it is a software value depreciation rate.
- *Number of uses.* This parameter is a number of uses that the human users plan to use this software component in their software applications within a time-unit of the length of planning horizon.
- *Maintenance cost.* This value indicates maintenance cost for the software component once it is bought within one time-unit of the length of planning horizon.
- *Prefer price.* This is the price that the human user would like to pay for the software component. It is optional, but once this parameter is set by the human user, this value will be user agent's first trading price for buying or selling the software component.
- *Acceptable price.* This price is the minimal or maximum price that the human user can accept. For buying or renting a software component, this price is the highest price that software vendor is willing to pay. For selling, this price is the lowest price that software vendor is willing to sell the software component.

- *Time window.* This value is a time period during which the buyer agent and seller agent can trade the software component in the marketplaces. After this time window expires, the buyer or seller must finish trading and come back to report the trading result.
- *Sub-components.* This parameter indicates how many sub-components have been used to build the software component that will be selling in the markets. That parameter will be used to calculate the total cost of the software component.

Among these parameters, *the length of planning horizon, discounting interest rate, number of uses and maintenance cost* will be four key parameters used in our buy/rent decision algorithm. The *sub-components* is an only parameter used in when to sell decision algorithm.

Following Figure 4.5 shows one instance of the User agent GUI after the human user input all the values for the parameters to require a software component – in this case, it is Jsockets. This component is a Java software component which can be used to create HTTP-tunneling socket for software applications. We use it as an example in this thesis.

The screenshot shows a GUI window titled "user1" with standard window controls (minimize, maximize, close) in the top right corner. The window contains several input fields with labels and corresponding values:

- Enter The Software Component: jsockets
- Enter the length of the planning horizon (in time units): 4
- Enter current Discounting Rate: 0.3
- Enter The number of Usage in a time unit: 20
- Enter maintenance cost for this software component: 10
- Enter a prefer price for this software component: 90
- Enter an acceptable price for this software component: 100
- Enter The time window for This trade (TimeOut): 8
- Type in sub-components used for this component: (empty field)

Below the input fields are two buttons: "Send Buyer Agent" and "Send Seller Agent". Underneath these buttons is the text "User Agent Searched Result". At the bottom of the window is a scrollable text area containing the following text:

Activate User Agent.....
Send BuyAgent to buy jsockets at price : 90.0

Figure 4.5 GUI Picture of Decision-making to buy by User Agent

We can see that the user agent has made decision to buy jsockets at price 90. The following Figure 4.6 is illustrating the trading result for buying jsockets activity.

The screenshot shows a window titled "user1" with the following fields and values:

- Enter The Software Component: jsockets
- Enter the length of the planning horizon (in timeUnits): 4
- Enter Current Discounting Rate: 0.3
- Enter The number of Usage in a timeUnit: 20
- Enter maintenance cost for this software component: 10
- Enter a prefer price for this software component: 90
- Enter an acceptable price for this software component: 100
- Enter The time window for This trade (TimeOut): 8
- Type in sub-components used for this component: (empty)

Below the input fields are two buttons: "Send Buyer Agent" and "Send Seller Agent".

The "User Agent Searched Result" section displays the following information:

```

Activate User Agent.....

Send BuyAgent to buy jsockets at price : 90.0
-----
Bought Software Component : jsockets
Seller CompanyID: CA55555

Buyer Agent Paid Price: 91.0
Total Money Left: 9909.0
Market Average Price is: 98.19
-----

```

Figure 4.6 Buying Trading Result Picture of The User Agent.

From Figure 4.6, we can see that a jsockets has been bought from the marketplaces at price 91.0. The seller's information and current market average price for the jsockets are also displayed.

The following Figure 4.7 is illustrating the decision has been made by the user agent to rent a jsockets from marketplaces and trading result from renting activity.

The screenshot shows a GUI window titled "user 1" with the following fields and values:

- Enter The Software Component: jsockets
- Enter the length of the planning horizon (in timeUnits): 4
- Enter Current Discounting Rate: 0.3
- Enter The number of Usage in a timeUnit: 1
- Enter maintenance cost for this software component: 10
- Enter a prefer price for this software component: 90
- Enter an acceptable price for this software component: 100
- Enter The time window for This trade (TimeOut): 8
- Type in sub-components used for this component: (empty)

Below the input fields are two buttons: "Send Buyer Agent" and "Send Seller Agent". Below these buttons is a section titled "User Agent Searched Result:" containing a text area with the following text:

```

-----
Activate User Agent....

Send BuyAgent to rent jsockets at price : 12.0|
-----
Renting Software Component : jsockets
No Supplier Found

Total Money Have is 9909.0
Market Average Renting Price is: 26.62
-----

```

Figure 4.7 GUI Picture of Decision-making to rent by User Agent

From Figure 4.6 & 4.7, we can notice value of the parameter *number of uses* is different, this difference makes the user agent has different decisions made for the human user (one is to buy and another is to rent). This decision algorithm has be explained in section 3.2.2. The detail implementation of this decision-making algorithm will be illustrated in next section.

For the human user who likes to sell a software component - jsockets, the following Figure 4.8 illustrates the selling process and trading result.

The screenshot shows a window titled 'user2' with the following fields and values:

- Enter The Software Component: jsockets
- Enter the length of the planning horizon (in timeUnits): 0
- Enter Current Discounting Rate: 0
- Enter The number of Usage in a timeUnit: 0
- Enter maintenance cost for this software component: 0
- Enter a prefer price for this software component: 93
- Enter an acceptable price for this software component: 90
- Enter The time window for This trade (TimeOut): 8
- Type in sub-components used for this component: (empty)

Below the input fields are two buttons: 'Send Buyer Agent' and 'Send Seller Agent'. Below these buttons is a section titled 'User Agent Searched Result' containing the following text:

```

Activate User Agent.....

Send SellAgent to sell jsockets at price : 93.0
lease at price : 31.0

-----
Software Component : jsockets
Seller Agent Sold out 1 copies
Seller Agent Lease out 1 copies
Seller Agent Made Money : 121.0
Total Money The User Has Now is : 10121.0

Market Average Sell Price is: 92.97
Market Average Lease Price is: 31.2
  
```

Figure 4.8 Selling Trading Picture of The User Agent

From Figure 4.8, we can see that the user agent decided to sell this jsockets at 93.0 and lease it at 31.0 in the marketplaces, and trading result is that the seller agent has sold one copy and leased one copy at the user agent's prefer price. We can notice that the total earned money for the user agent is less than the total sale and lease money, this is because a certain amount of money is paid to market agents as commissions. To understand these

trading processes, we will present user agent's detail design and implementation in following section.

4.4.1.1 Design and Implementation of User Agent

As we stated in section 3.1 of this chapter, the Tryllian agent's behavior is controlled by a composition of tasks. For user agents, a number of tasks have been implemented in Java classes based on ADK. They are UserGUITask, SendAgentTask, HandleMessageTask, FindLocationTask and CreateConfedAgentTask. Following UML class diagram illustrates the task

classes in user agent.

Figure 4.9 UML Class Diagram of The User Agent

Following Figure 4.10 illustrates task structures of the user agent.

Figure 4.10 Tasks workflow Diagram of The User Agent

Following diagram illustrates state diagram of the user agent.

Figure 4.11 State diagram of the user agent

In Figure 4.10 and 4.11, we can see that the user agent uses a number of task schedulers to model its behaviors. Task scheduler 1, Task scheduler 2 and **HandleMessageTask** are running in parallel in the user agent. Task scheduler 1 contains **FindLocationTask** and **CreateConfedAgentTask.Task**

scheduler 2 contains **UserGUITask** and **SendAgentTask**, The Figure 3.12 illustrates a sequence diagram of the user agent's trading activities.

Figure 4.12 Sequence Diagram of User Agent's Tasks.

From diagram 4.12, we can see that the user agent starts by executing task scheduler 1 to create a number of confederation agents to discover

marketplaces' locations in networks. A parallel running **HandleMessageTask** will receive all the reporting messages from confederation agents on the subject of **INFO_SUBJECT**. The **HandleMessageTask** will use this information to build a marketplaces database for the user agent. This marketplace location database gives the user agent the knowledge on where to trade regarding on each specific software components, the database is implemented using a Hashtable data structure in which each software component is key and a list of corresponding marketplaces is value. The marketplace discovery algorithm in confederation agent will be introduced in confederation agent section 4.2.2.

The task scheduler 2 that contains **UserGUITask** and **SendAgentTask** will constantly run in a loop within **UserGUITask** to wait for human user's input. Once the human user decides to buy or sell a software component by click action buttons – Send Buyer Agent or Send Seller Agent, **SendAgentTask** will be invoked to create a query agent which will go to marketplaces to query current average prices for buying and renting the software component. This query agent will report the market average price to the user agent by sending message with subject of **AVEPRICE_SUBJECT**. The **HandleMessageTask** is a task that will handle all the messages sent to the user agent. And also inside the **HandleMessageTask**, a business decision-making function is implemented based on a business-making model that we described in chapter 3, this decision function will make buying/renting or when to sell

decision based on the market prices received and parameter that the human user input. If the human user provides a preferred price for the component, then it will be used as the price for this software component. In case when the preferred price is not provided, to make this simulation market close to real marketplace, this decision function will decide an initial offering price based on current market average price within a random range. The function is implemented in following Java code:

```
private void decisionMaking(){

String action = (String) getAgentContext().getProperty("SellOrBuy");
.....
if(action.equals("BuyAgent")){
    String type = "";
    String sPrice = "";
    double bid = 0;
    //check the market prices that we have
    if( marketSellPriceForSW != 0 && marketRentPriceForSW != 0)
    {
        double rentAnnualCost = marketRentPriceForSW*numberOfUse;
        //pass into a present function to calculate the present value of total cost
        double totalCostForRent = presentValue(rentAnnualCost, interest,
length);

        double totalMaintainCost = presentValue(maintenanceCost, interest,
length);

        double totalCostForBuy = marketSellPriceForSW + totalMaintainCost;

        //now compare the costs of rent and buy
        if(totalCostForBuy < totalCostForRent)
        {
            type = "buy";
            if(preferPrice != 0)
                bid = preferPrice;
            else
                bid = marketSellPriceForSW(1 - Math.random()/10);
        }
    }
    else{
```

```

        type = "rent";
        bid = marketRentPriceForSW(1 - Math.random()/10);
    }

}
else {
    type = "buy";
    bid = preferPrice;
}

sPrice = Double.toString(Math.floor(bid));
//set the value int agent's context
getAgentContext().setProperty("TypeOfAction", type);
getAgentContext().setProperty("BuyComponentPrice", sPrice);
}
else if(action.equals("SellAgent")){

    String sellPrice = "";
    String rentPrice = "";

    if(totalSubComponentsSell < marketSellPriceForSW ){

        double cost = totalSubComponentsSell;
        //for sell the product, it is to decide price based on the market
        if( marketSellPriceForSW != 0 && marketRentPriceForSW != 0 )
        {
            if(preferPrice != 0)
                sellPrice = Double.toString(preferPrice);
            else
                sellPrice
                Double.toString(Math.floor(marketSellPriceForSW(1 + Math.random()/10)));
                =

                rentPrice
                Double.toString(Math.floor(marketRentPriceForSW(1 + Math.random()/10)));
                =

        }
        else
        {
            if(preferPrice != 0 )
                sellPrice = Double.toString(preferPrice);
            else
                sellPrice
                Double.toString(Math.floor(marketSellPriceForSW(1 + Math.random()/10)));
                =

                rentPrice = Double.toString(Math.floor(0.2*(marketSellPriceForSW(1
+ Math.random()/10))));

        }

    }

}
//develop cost great than possible selling price
else {

```

```

        sellPrice = "0";
        rentPrice = "0";
    }
    //set the values into agent's context
    getAgentContext().setProperty("SellComponentPrice", sellPrice);
    getAgentContext().setProperty("RentComponentPrice", rentPrice);
}
getAgentContext().setProperty("DecisionMakeFinished", "true");
}

```

After the user agent made decision by using the above function, the **SendAgentTask** in the user agent will create a buyer agent or a seller agent to go to the marketplaces to trade the software component. After the buyer agent or the seller agent finished their trading mission, a trading result will be reported to the user agent through **HandleMessageTask** by subjects of BUY_SUBJECT or SELL_SUBJECT, then displayed to human user (software component vendors) by **UserGUITask**.

In order to shorten trading time for buyer agent or seller agent, a preference marketplaces selection algorithm has been implemented in **SendAgentTask**. By using this algorithm, the buyer agent or seller agent will be given an itinerary of marketplaces in which the preferred marketplaces will be placed at beginning of the itinerary. So the buyer agent or the seller agent will visit these markets first in a hope to find a deal quickly. This preferred marketplaces database is built by the user agent based on its previous trading experience. Each time when the user agent procured or sold a software component in a marketplace, the user agent will put this marketplace

into its preferred markets database. So next time when the user agent needs to buy or sell the same type of the component, this market will be considered as a preferred marketplace to go.

This market selection algorithm is implemented in following Java function:

```
private void setPreferMarketAddresses(String mAddrs, String component){
    Hashtable preferMarkets = (Hashtable)
        getAgentContext().getProperty("preferMarketList");

    if(preferMarkets.containsKey(component)){
        //get the market list
        Vector v = new Vector();
        String aList = (String) preferMarkets.get(component);
        StringTokenizer st = new StringTokenizer( aList, ",");

        while (st.hasMoreTokens()) {
            String ad = st.nextToken().trim();
            v.addElement(ad);
        }

        //we use the random function to decide the next first market to
        //go, that will increase the agent flexibility to reach the varied
        //market for better chance to find a match.
        String firstMarket = "";
        double ran = Math.random();
        int s = v.size();

        if(s == 1)
            firstMarket = (String) v.get(0);
        if(s == 2)
        {
            if (ran >= 0 && ran < 0.5)
                firstMarket = (String) v.get(0);
            else
                firstMarket = (String) v.get(1);
        }

        if(s >= 3)
        {
            if(ran >= 0 && ran < 0.5)
                firstMarket = (String) v.get(0);
            else if(ran < 0.8 && ran >= 0.5)
                firstMarket = (String) v.get(1);
            else
                firstMarket = (String) v.get(2);
        }
    }
}
```

```

    }

    //now we examine the mAddrs for the markets that agent will visit
    Vector addressVector = new Vector();
    StringTokenizer tokens = new StringTokenizer( mAddrs, ",");
    while (tokens.hasMoreTokens()) {
        String adr = tokens.nextToken().trim();
        addressVector.addElement(adr);
    }//end while

    //if it is in the list then we want to move it to first position

    if(addressVector.contains(firstMarket))
    {
        addressVector.remove(firstMarket);
        mAddrs = firstMarket;
        for(int i=0; i<addressVector.size(); i++)
        {
            mAddrs += "," + (String) addressVector.get(i);
        }
    }
}

```

In the above implementation, in order not to lock our preferring market choice to a fixed marketplace in our preferring marketplace database, we use a random number to decide which marketplace will be our preferring marketplace to go. To simplify this selection function in this thesis, we only keep the maximum of three marketplaces in our preferring market database to reflect more recent trading activities. We use random number range within 0 to 0.5 as our testing condition to choose the first market as our preferring marketplace, 0.5 to 0.8 to the second marketplace, 0.8 to 1 to the third marketplace.

3.4.2 Confederation Agent

Confederation agent is an agent that is designed to discover marketplaces' locations in wide spread networks. This markets discovery algorithm is based on [7]. The idea is to use mobile confederation agents to move around in networks, and exchanging the information (marketplace addresses) with every marketplace node that they encounter on their route. After a pre-set time period expires, the confederation agents will come back to the originated user agent to report all the marketplace addresses that they have acquired, then the user agent can build a software components marketplace addresses database.

The discovery algorithm of the confederation agent is as follows:

1. First of all, a number of confederation agents will be created by a user agent. These confederation agents update themselves with address information of its creator (the user agent), and also put the address of this creator as their home address. A journey-time for the confederation agent is set in terms of the maximum number of nodes that may be visited before it must return to its creator. Also a branching factor is set, that is used to determine how many times the mobile agent may be copied at any one node in the networks.

2. Initially each confederation agent is given addresses of some other nodes participating in the market network. These addresses constitute first itinerary of each confederation agent. Typically the number of addresses in this itinerary is small, and less than the branching factor.
3. After arriving at each node the confederation agent decrements its journey time, and updates the node with information (address) about its creator, and other marketplaces encountered along the route so far. The confederation agent also updates itself with information (address of this marketplace and what software components this market is able to trade) of the current node (marketplace).
4. If the confederation agent's journey-time has expired, then it returns to its creator (the user agent), and updates the user agent with all the information it has collected on its journeys. Otherwise the confederation agent copies itself enough times to allow a copy of itself to be sent to other market places that was known to the current market but unknown to the confederation agent. By excluding previously known marketplaces, the incidence of revisits is reduced.
5. The process is continued from step 3.

Because of time window (pre-set journey time) constrain, this algorithm is not designed to discover a total and complete marketplaces in all large networks. Rather it can provide a relative comprehensive list of marketplace addresses and detail information on feasible trading software components for the user agent. This marketplace information is stored in the confederation agent and the user agent using a Hashtable data structure:

- Name of software component as key.
- The list of marketplaces' address that is able to trade this software component as value.

The confederation agent achieves its behavior to move and communicate with marketplaces by using a task scheduler in which a sequence of subtasks is executed in a pre-defined logic order. A task scheduler is a special type of task that can be used to organize a set of sub-tasks in logic manner. The following Figure 4.13 illustrates the task sequence of the confederation agent.

Figure 4.13 Task Sequence Diagram of The Confederation Agent

From the Figure 4.13, we can see the sub task FindLocationTask will be executed first to find out the initial location of marketplace to go. Then CheckFinishedTask will be invoked to check if the time is up, if not, it goes to MoveToLocationTask to move to the known marketplace, if yes, the agent will execute ReturnHomeTask to return to its home location. After the

confederation agent move to the marketplace, it will use AskInfoTask to communicate with the market agent using INFO_SUBJECT, so that the confederation agent can find out the address of the market agent. If the confederation agent found new addresses information(new nodes), it will copy itself for a number of times to send to the new addresses, and decrement its time limit. Then the confederation agent will execute CheckFinishedTask to see if the time is up. If no, goes to new location and repeat previous steps again, if yes, the confederation agent will return to its home location to report its marketplace information to the user agent. This task scheduler is implemented in following Java code:

```
// Create a task scheduler
TaskScheduler ts = new TaskScheduler();

// The format of adding subtasks to a scheduler is:
// addtask(scheduled_task, next_task_on_succeed, next_task_on_fail);

ts.addTask(findLocationsTask, checkFinishedTask, reportTask);
ts.addTask(checkFinishedTask, returnHomeTask, moveToLocationTask);
ts.addTask(moveToLocationTask, askInfoTask, checkFinishedTask);
ts.addTask(askInfoTask, checkFinishedTask, moveToLocationTask);
ts.addTask(returnHomeTask, reportTask, null);
ts.addTask(reportTask, dieTask);
ts.addTask(dieTask, null, null);

// Set the starting task of the scheduler
ts.setFirstTask(findLocationsTask);
```

Every confederation agent will destroyed itself after it finished its mission to give back the memory resource to the computer system.

4.4.3 Query Agent

Query agent is an agent created by the user agent on purpose of querying current market prices for buying and renting a software component. The query agent plays an important role in decision making process, because the average prices of buying and renting the software component will be used in our business decision function to calculate the total cost of buying and renting the software component.

A query agent will be created at the beginning of the user agent's decision making process. Similar to confederation agent, Its behavior is also controlled by a set of tasks that are scheduled by a similar task scheduler. Figure 4.14 illustrated the task sequence in query agent's task scheduler. The query agent will also destroy itself after its task finished.

Following Figure 4.14 shows UML class diagram of the query agent.

Figure 4.14 UML class diagram of the query agent.

The sub-tasks in query agent's task scheduler is arranged in a logic sequence as shown in following code:

```

TaskScheduler ts = new TaskScheduler();
ts.addTask(findLocationsTask, checkFinishedTask, reportTask);
ts.addTask(checkFinishedTask, returnHomeTask, moveToLocationTask);
ts.addTask(moveToLocationTask, askInfoTask, checkFinishedTask);
ts.addTask(askInfoTask, checkFinishedTask, moveToLocationTask);
ts.addTask(returnHomeTask, reportTask);

```

```

ts.addTask(reportTask, dieTask);
ts.addTask(dieTask, null, null);
// Set the starting task of the scheduler
ts.setFirstTask(findLocationsTask);

```

Following Figure 4.15 illustrates this logic sequence of sub-tasks in task scheduler.

Figure 4.15 Task Sequence Diagram of the Query Agent

From Figure 4.15, we can see that, after a query agent is created by the user agent, its **FindLocationsTask** will find marketplace addresses that the user agent provided to build the query agent's itinerary. If there are some marketplaces in its itinerary, the query agent will go to execute

CheckFinishedTask, otherwise it will return home by calling ReturnHomeTask.

Then **CheckFinishedTask** will check whether the query agent has found average market prices for a particular software component. If the prices are found, the query agent will return to its home place - its creator's place (the user agent's place) by executing returnHomeTask, otherwise call moveToLocationTask to move to next marketplace to query for market prices for the software component.

After **MoveToLocationTask** moved the query agent to next marketplace, **AskInfoTask** will send a price query message with subject of AVEPRICE_SUBJECT to market agent on the software component, then wait for reply from the market agent. When AskInfoTask gets a reply message from the market agent, this task will succeed. The AskInfoTask will fail when there is timeout or it did not receive any reply. When it succeeds, the query agent will go to execute CheckFinishedTask, otherwise executes MoveToLocationTask.

ReturnHomeTask will move the query agent back to its home location in where it was created. The user agent will execute reportTask when it returns home. Otherwise the query agent couldn't return home, error message will be provided to human user.

ReportTask will report the current average price information on buying and renting the software component to the user agent with a message subject of AVEPRICE_SUBJECT. This price information is important for decision-making function of the user agent to make decision on whether to buy or rent the software component. After ReportTask is executed, the query agent will execute the DieTask.

DieTask will destroy the query agent to give memory space back to the compute system.

4.4.4 Buyer Agent

Buyer agent's duty is to go to marketplaces to buy or rent the software component at certain price range on behalf of the user agent.

4.4.4.1 Buyer Agent's Parameters

We will first take look at the parameters that are passed in by the user agent at the time when the buyer agent was created. Following construction code in buyer agent class illustrates these parameters:

```
public BuyAgent(String[] args) {  
    // At the time when user agent creates a buyer agent,  
    // an array of parms will passed in.  
    if (args != null && args.length > 0) {
```

```

        setProperty("buySWComponent", args[0]);
        setProperty("MarketAgentAddrs", args[1]);
        setName(args[2]);
        setProperty("buyComponentPrice", args[3]);
        setProperty("TypeOfBuy", args[4]);
        setProperty("TimeOut", args[5]);
        setProperty("companyID", args[6]);
        setProperty("AcceptablePrice", args[7]);
    }
    else {
        setName("BuyAgent");
        System.err.println("ERROR - could not set parms for buy agent!");
    }
}

```

From the above code, we can see that buyer agent initialize a number of its internal parameters using the arguments passed in by the user agent.

They are as follows:

BuySWComponent -- this is name of the software component that the user agent likes to buy or rent in marketplaces.

MarketAgentAddrs -- this value contains a list of marketplace's addresses by which the buyer agent can construct its itinerary in networks. The first marketplace of this list is usually a user's preferring marketplace.

BuyAgentName -- this value is a unique name of the buyer agent, which is created by the user agent to ensure this buyer agent can be identified in the networks.

BuyComponentPrice -- this price is calculated by the user agent's decision-making function that we have illustrated in the user agent's section. It is a user agent's prefer price to buy or rent the software component in the markets.

AcceptablePrice – this price is the highest price that the software component vendor can accept when the buyer agent buys or rents the software component in the marketplaces.

TypeOfBuy -- this parameter tells the buyer agent which action (buying or renting) to perform in the marketplaces. It is decided by the user agent's decision-making function.

TimeOut -- this value is time window value that the human user input in the user agent's GUI window. This timeout period constrains the trading time in which the buy agent can live.

CompanyID – it is the human user's company's ID in marketplaces.

All of the above parameter values are either calculated or set by the user agent and the human user. With these values, the buyer agent can carry out a buying or renting mission for the software component in a number of marketplaces.

4.4.4.2 Detail Function Design of Buyer Agent

The buyer agent has the same task scheduler structure and UML class structure as the query agent that are shown in Figure 4.15 & 4.16. Please refer to these two Figures for the buyer agent's task structures. The following

Figure 4.16 is a sequence diagram to illustrate how the buyer agent trades in a marketplace, Figure 4.16 is state diagram of the buyer agent.

Figure 4.16 Message Sequence Diagram of The Buyer Agent

Figure 4.17 State Diagram of The Buyer Agent

As we can see from Figure 4.16 & 4.17, a task scheduler controls the behavior of the buyer agent. First, **FindLocationTask** will build the user agent's itinerary by using the parameter – `MarketAgentAddrs`. If succeed, then the buyer agent will go to execute **CheckFinishedTask**, otherwise go to **reportTask**.

CheckFinishedTask will check whether the buy agent has bought or rented a software component that the user agent is looking for, and whether the time window has expired. If the software component is found or the time window

has expired, then the buyer agent goes to execute **ReturnHomeTask**, otherwise executes **MoveToLocationTask** to move the buyer agent to next marketplace to continue trading.

MoveToLocationTask will move the buyer agent to a marketplace. If succeed, then executes **AskInfoTask**, otherwise, **CheckFinishedTask**.

In **AskInfoTask**, the buyer agent will send a message to a market agent to buy or rent the software component on the subject of **BUY_SUBJECT**. The message sent by the buyer agent contains **BuyAgentName**, **BuySWComponent**, **TypeOfBuy** and **BuyComponentPrice**. After the market agent processed this message, the market agent will send a reply message to the buyer agent. The reply message contains information on the market agent's address, whether the requested software component has been found and current market average price for this software component. If the software component is found, the seller's companyID will be included in the reply message. If the buyer agent successfully bought or rented the software component, then go to execute **CheckFinshedTask**. Otherwise the buyer agent will remember this marketplace address and adjust its corresponding buying or renting price so that next time when the buyer agent re-visits this marketplace, it will send a new adjusted price to this marketplace agent in order to increase its chances for finding a match. And then executes **MoveToLocationTask** to move to next marketplace to trade for the software

component. The price adjust algorithm is achieved by following Java code in the **AskInfoTask** and **MoveToLocationTask**.

```
//in AskInfoTask
if (Constants.Subjects.BUY_SUBJECT.
    equals(incomingMessage.getSubject()))
{
    log("Incoming reply message from InfoAgent.");
}
else
{
    log("Unexpected Subject of incoming message : " +
        incomingMessage.getSubject());
}

....
Public void handleMessage(IncomingMessage incomingMessage) {
.....
.....
//market address in reply message
String marketAgentAdd = incomingMessage.getContent(MARKETAGENTVISITED);

if(marketAgentAdd != null)
{
    try {
        PlaceAddress infoadd =
            (PlaceAddress)PlaceAddress.create(marketAgentAdd);
        Vector visitedMarketagents =
            (Vector)getAgentContext().getProperty("MarketAgentVisited");

        //this hashtable remember the marketplaces and the corresponding new
        //adjusted prices for them.
        Hashtable pricesAdjustedOnMarkets = (Hashtable)
            getAgentContext().getProperty("pricesForVisitedMarket");

        //we didn't find a match for this software component
        if(softInfo.containsValue(null))
        {
            //if we didn't find a match product, we will linearly increase
            //offering price by 2% to increase the chance for next revisit.
            String bidPrice = (String)
                getAgentContext().getProperty("buyComponentPrice");
            double newPrice = Double.parseDouble(bidPrice)*(1.02);

            //check if this new price has exceeded the highest Price that the user
            //can accept for this software component.
            if(newPrice <= highestPrice) {
```

```

        pricesAdjustedOnMarkets.put(marketAgentAdd,
                                   Double.toString(newPrice));
//set this table into agent's internal memory
getAgentContext().setProperty("pricesForVisitedMarket",pricesAdjustedOnMarkets);

    }

    ...
}

//in MoveToLocationTask
public void taskStarted() {
    .....

    // Get the next location
    int index = Integer.parseInt((String) getAgentContext().
        getProperty(NEXT_LOCATION));
    PlaceAddress location = (PlaceAddress) locations.get(index);

    String nextAdd = location.toExternalFormat();

    //get hashtable that contains all the visited marketplace and its
    //corresponding prices.
    Hashtable pricesOnVisitedMarkets = (Hashtable)
        getAgentContext().getProperty("pricesForVisitedMarket");

    //check to see if we have visited this market, if it is, then we set
    //its new price for this market.
    if(pricesOnVisitedMarkets.containsKey(nextAdd))
    {
        String newPrice = (String) pricesOnVisitedMarkets.get(nextAdd);

        getAgentContext().setProperty("buyComponentPrice", newPrice);
    }

    ....
}

```

ReturnHomeTask will move the buyer agent back to its home location in where it was created by the user agent.

ReportTask is a task that the buyer agent executes to report the trading result to the user agent after the buyer agent came home. This task will not succeed until it receives a confirmation message from the user agent, then goes to execute **DieTask**

DieTask will destroy the buyer agent give back the memory space to computer system.

4.4.5 Seller Agent

A seller agent is another mobile agent that is created by a user agent to sell on behalf of the user agent in marketplaces. The following parameters are passed into the seller agent by the user agent when the seller agent is created.

SellSWComponent -- this is name of a software component that the user agent would like to sell and lease in marketplaces.

MarketAgentAddrs -- this is a list of marketplace addresses by which the seller agent can build its itinerary in marketplaces network. As we have illustrated preferred market selection algorithm in the user agent section, the first marketplace of this list is usually the user agent's preferred marketplace.

SellAgentName -- this is a unique name of the seller agent that is created by the user agent in order to identify the seller agent in the market networks.

SellComponentPrice -- this price is the user agent's desired price for selling the software component. It could be adjusted during trading activities of the seller agent. This price adjustment algorithm will be introduced in late of this section.

AcceptablePrice – this price is the lowest price that the human user can accept when the seller agent sells the software component in marketplaces.

RentComponentPrice -- this rent price is the user agent's prefer price for leasing out the software component. It can also be adjusted during trading.

TimeOut -- this is a time window period during which the seller agent can do selling in markets, when it expires, the seller agent must return to its home location to report the trading result to the user agent.

CompanyID – it is the human user's company's ID in marketplaces.

All of the above parameter values are calculated and set by the user agent or human user.

4.4.5.1 Functional Design of Seller Agent

Seller agent has the same task scheduler structure and class relations as illustrated in Figure 4.14 and Figure 4.15 for the query agent. When a software component is leasing as a service or sold as an information good, they are all needed support and maintenance effort at seller's side. These conditions could affect the number of copies the seller can sell and lease in a marketplace, this is another complex business issues for the sellers which is

outside of our research scope. In this thesis, to simply our discussions, we design every seller agent can only sell and lease one copy of the software component in each marketplace. The following Figure 4.17 shows the message sequence diagram for the seller agent.

Figure 4.18 message sequence diagram of the seller agent.

From Figure 4.18, we can see the seller agent has the similar message sequence as Figure 4.16 shown for the buyer agent, except following differences.

- In **CheckFinishedTask**, the seller agent will only check if the time window has expired while the buyer agent will check whether the component has been bought or the time window has expired. That means the seller agent will sell its software component into many marketplaces within its time window. Whereas the buyer agent will return to its home location as soon as it bought or rented a software component within the time window. For the seller agent, once the time window is expired, this task will replace the seller agent's itinerary with the list of visited marketplaces in which the seller agent hasn't sold or leased the software component. So that the seller agent can clear all the remaining selling requests in these markets.
- The seller agent's message to the market agent contains two prices - the one is a selling price, another one is a renting price. While the buyer agent's message to the market agent contain its preferring buying or renting price and a action type (buy or rent). And the selling message form the seller agent also contains the seller's companyID.

The same as the buyer agent, the seller agent will remember the visited marketplaces for its selling prices adjusting purpose in those markets. The seller agent's price adjustment function is the same as the buyer agent except the seller agent decreasing the selling price linearly, while the buyer agent

increases the buying or renting price linearly, this function has been shown in the buyer agent's section 4.2.4.2.

The seller agent's clearing function for the remaining selling requests is explained in following code:

In SellInfoTask:

```
public void handleMessage(IncomingMessage incomingMessage) {  
  
    if (Constants.Subjects.SELL_SUBJECT.  
        equals(incomingMessage.getSubject()))  
    {  
        log("Incoming reply message from InfoAgent.");  
    }  
    else  
    {  
        log("Unexpected Subject of incoming message : " +  
            incomingMessage.getSubject());  
    }  
    .....  
    ...  
  
    String marketAgentAdd = incomingMessage.getContent(MARKETAGENTVISITED);  
    if(marketAgentAdd != null)  
    {  
        try {  
            PlaceAddress infoadd = (PlaceAddress)  
                PlaceAddress.create(marketAgentAdd);  
            Vector visitedInfoagents =  
                (Vector)getAgentContext().getProperty("MarketAgentVisited");  
  
            String timetoclear = (String)  
                getAgentContext().getProperty("TimeToClear");  
  
            //check if it is timeout occurred  
            if(timetoclear.equals("true"))  
            {  
                if(visitedInfoagents.contains(infoadd))  
                {  
                    visitedInfoagents.removeElement(infoadd);  
                }  
            }  
        }  
    }  
}
```

```

        getAgentContext().setProperty("MarketAgentVisited",
visitedInfoagents);
    }

    }
    else {
        // add the visited marketagent to the vector as it will be use to
        //clear selling requestes
        if(!visitedInfoagents.contains(infoadd))
        {
            visitedInfoagents.addElement(infoadd);
            getAgentContext().setProperty("MarketAgentVisited",
visitedInfoagents);
        }
    }
    .....
}

```

In CheckFinishedTask:

```

public void taskStarted() {

    getAgentContext().getProperty("TimeOut");

    if (timeout > 0)
    {
        timeout--;
        getAgentContext().setProperty("TimeToClear", "false");
        getAgentContext().setProperty("TimeOut", Integer.toString(timeout));
    }
    else
        getAgentContext().setProperty("TimeToClear", "true");

    if (timeout != 0) {

        msg += "still selling the product";

        // Get the locations itinerary
        Vector locations = (Vector) getAgentContext().
        getProperty(LOCATIONS);

        // If there are no locations to visit, succeed this task
        // (agent is finished)
        if (locations == null || locations.size() == 0) {

            msg += "no locations available";
            succeed();
        }
        // If there is a location to visit, fail this task
        // (agent not yet finished)
        else {

```

```

        msg += "will try the next location";
        fail();
    }
}

// otherwise the timeout and we need to clear selling requests
else {

    Vector visitedinfoAgents = (Vector)
        getAgentContext().getProperty("MarketAgentVisited");
    //we are done, there is no market to clear our requests
    if(visitedinfoAgents.isEmpty())
    {
        succeed();
        msg += "Finish sell software component";
    }
    else {
        //we reset the LOCATIONS(itinerary) for seller agent
        // to clear all the selling requests
        getAgentContext().setProperty(LOCATIONS, visitedinfoAgents);

        getAgentContext().setProperty(NEXT_LOCATION, "0");

        getAgentContext().setProperty("TimeToClear", "true");

        fail();
    }

}

}

```

- The content of trading result from the seller agent is different from the buyer agent. While the buyer agent will report the transaction price, seller's companyID and current market average price, the seller agent will report how many copies sold and rented, the total money made and current market average price and current average price.

3.4.6 Market Agent

A market agent is a stationary agent who manages a marketplace in which sellers and buyers can trade their software components. Every marketplace has only one market agent. Every market agent provides a GUI interface window to demonstrate the market activities (buying/selling) and market current status. The following two Figures 4.19 & 4.20 show a marketplace during trading and after trading.

Figure 4.19 GUI Picture of Trading Activity of The Market Agent.

Figure 4.20 GUI Picture of Trading Statistic Result of The Market Agent

From Figure 4.19 & 4.20, we can see the first line in the GUI is illustrating what software components this marketplace is able to trade, in this example, they are jsockets and inetfactory. The second line in the GUI is displaying what software component is currently trading in this marketplace, in this case, it is inetfactory. And next few lines are displaying how many buyers and sellers in this marketplace and what price they are offering. Then the market earned commission and average selling and renting price will be displayed. In Figure

4.20, after the marketplace finished all the trading on inetfactory, it will display some statistic data for inetfactory such as total number of buyers and sellers, how many of them have been matched.

Following Figure 4.21 and 4.22 illustrate the market agent's task structure and class relations.

Figure 4.21 Task structure of the market Agent

Figure 4.22 UML Class Diagram of the Market Agent.

As shown in Figure 4.21 and 4.22, **HandleMessageTask** and **MarketGUITask** are running in parallel within the market agent. There is no logic sequence between them. In order not to interfere with the **HandleMessageTask**, the **MarketGUITask** has been designed to run in a separate thread.

The **MarketGUITask** creates a marketplace window. This task has two main functions `start()` –create a GUI window, `refreshGUI()` – refresh the window every time the marketplace activities happened. This `refreshGUI()` function will be called by the **HandleMessageTask** every time there is a trading activity in the marketplace.

HandleMessageTask is a task that handles all the messages that are sent to the market agent from confederation agent, query agent, seller agent and buyer agent. Following is a detail description of functionality that the **HandleMessageTask** provides.

As requirements of this software component e-marketplace that are described in section 2.3, each marketplace must be able to trade a number of different types of software components, one type of software component can be traded in a number of different marketplaces. To fulfill this requirement, the market agent is designed switch trading context among multiple software components in the **HandleMessageTask**. The trading context is a session in which one type of the software component is being traded by the market

agent. At any time, the market agent only can trade one type of the software components, this is because the ADK only supports single thread computation. Therefore, when the market agent receives many messages from different buyer agent and seller agent to ask for different types of software components, the market agent will have to store these requests in its buffer, and deal with them one by one. Before the market agent switches a trading context from one component to another, the market agent will use a number of intermediate files to store current trading information regarding on the software component such as current average price, current trader and statistic information. So that when the market agent switch back, it can continue trading what is left for this software component.

This **HandleMessageTask** deals with four types of the messages, the subjects of these message are **BUY_SUBJECT**, **SELL_SUBJECT**, **INFO_SUBJECT** and **AVEPRICE_SUBJECT** which has been explained in section 4.1.1. The follow Figure 4.23 is a message sequence diagram for the market agent.

Figure 4.23 Message Sequence Diagram of The Market Agent.

From Figure 4.23, the market agent's HandleMessageTask receives messages from confederation agent, query agent, buyer agent and seller agent.

- With a message from the confederation agent that is asking for address of the market agent and what software components the market agent can trade, the market agent will reply with its network address and a list of software components it can trade. The market agent will also use the network addresses of other nodes that are passed in by the confederation agent to build its address information database that contains the addresses of other marketplaces in networks. The function `dealConfedAgent()` is responsible for dealing with this message.
- With the message from query agent, the market agent will extract the average market prices from its corresponding software component's price file, and send this information to the query agent. The function `dealAveragePriceInquery()` will deal with this message.
- With the message from the seller agent, the market agent will store this selling software component, selling price, renting price, seller's companyID to its `master_ssnTable` and also put its selling price into its internal `sellPrices` (Hashtable). This list will be searched against a buyer's request. After these registrations, the market agent will reply with current market average prices and transaction status on whether the seller's component has been sold or not. The function `dealSellAgent()` will perform the above tasks.

- With the message from the buyer agent, the market agent will use the buyer's preferring price to search the corresponding seller's price list. If there is a match, then all the detail information (such as seller's companyID) of the seller will be passed over to the buyer, and the money will be transferred from the buyer to the seller. There will be a commission fee deducted from the money, the amount of this commission will be decided by each individual marketplace, which is initialized in the market agent creation stage. The function dealBuyAgent() will perform the above tasks.

This **HandleMessageTask** will constantly keep track of the number of the buyers and sellers in the marketplace and revise average prices for each software component.

4.5 Monetary, Security and Transaction System

As we stated in the scope and goal of this project, the focus of this research is not on banking, security and transaction system of the e-marketplace. So we use virtual money as a medium to deal with transactions among markets, buyers and sellers. Also we do not concern about the product transaction issue in this software component e-marketplace project. This transaction process will involve a complex law enforcement sub-system which is not in

our current scope of this research, so we just simply assume the buyer will contact the seller for the transaction time, services and the method of transaction. Regarding on security issue, it is also not our concern in this research, it could be a topic of our future research in this e-marketplace.

This chapter described a detail implementation of our e-marketplace research project. Next chapter will present a few of evaluation tests that are to verify the main functionality of this e-marketplace.

Chapter 5

Experimental Validation

In this chapter, we will demonstrate a few experimental tests of this e-marketplace. The section 5.1 is the functional testing of the user agent sub-system. The section 5.2 is the functional testing on Market Agent sub-system.

First of all, in this experimental test, we will create two e-marketplaces and two users using following XML file:

```
<!DOCTYPE habitat SYSTEM "http://www.tryllian.com/xml/habitat_1_0.dtd">
<habitat>
  <!-- <onException action="log"/> -->
  <places>

    <room name="/marketRoom1">
      <agent dna="agents/marketagent.dna">
        <arg value="Market1" />
        <arg value="0.05"/>
        <arg value="jsockets" />
        <arg value="2d/3dgraph" />
      </agent>
    </room>

    <room name="/marketRoom2">
      <agent dna="agents/marketagent.dna">
        <arg value="Market2" />
      </agent>
    </room>
  </places>
</habitat>
```

```

        <arg value="0.03"/>
        <arg value="jsockets" />
        <arg value="inetfactory" />
    </agent>
</room>

    <room name="/UserRoom1">
        <agent dna="agents/useragent.dna">
            <arg value="user1" />
            <arg value="Microsoft" />
        </agent>
    </room>

    <room name="/UserRoom2">
        <agent dna="agents/useragent.dna">
            <arg value="user2" />
            <arg value="SUN" />
        </agent>
    </room>

</places>

</habitat>

```

From the above file, we can see that two marketplaces are created in two separated rooms. Each marketplace is initialized with a number of values – the first one is the name of marketplace, second one is percentage of commission the marketplace will charge, the rest of two are types of software components this marketplace can trade. The number of the software components can be as many as the marketplace wants. Also there are two user agents created in two rooms as well, each user agent will be initialized with a user name and companyID. From the precious XML file, we can see that *jsockets* can be traded in both markets, and each market can trade

multiple software components, in this case, two software component in each marketplace.

Test Case 1. User Agent Marketplace Knowledge Validation

From the previous initializing XML file, we can see that in this e-marketplaces, there are three software component can be traded – *jsockets*, *2d/3dgraph* and *inetfactory*. In this test case, we will test whether the user agent will automatically learn the information of marketplaces' addresses and software components associated with them.

Testing input 1

We input software component name “*inetfactory*” into software component area of the user1's interface. Click send buy agent button.

Testing output 2

We saw the text area screen displays “*Activate the user agent...*”, this message indicates the use1 knows the address of the marketplace2. This is due to the confederation agents created by user1 have automatically discover the addresses of the marketplaces and their trading software components, and reported to the user1.

Test input 2:

We input a name “decisionTree” into the software component area of the user1's interface. Click send buy agent button.

Test output 2:

We saw that a message “There is no markets trading decisionTree, User agent action aborted.....” appears in result area of the user1’s GUI. This is due to the user1 has no marketplaces information associated with component decisionTree. From the previous XML file, we can check there is no marketplaces initialized to trade the *decisionTree*.

These two tests prove that the confederation agents are automatically discovering the marketplace information for the user agents.

Test Case 2. Parameter Sensitivity Testing on Buying/Renting Decision-making of User Agent

As we described in section 2.2.2, we used a total cost business decision model to calculate the total cost for buying and renting options, and make whether to buy/rent decision for the human user. This test is to evaluate sensitivities of all parameters related to this decision-making algorithm. We will test four parameters: *Length of planning horizon*, *discounting interest rate*, *number of uses* and *maintenance cost*.

To test the sensitivity of each parameter, we will fix the all other three parameters while we test one parameter. All the values we choose for these four parameters are arbitrary values, they have no special meanings.

- Length of planning horizon

First, let's look at *length of planning horizon*, this parameter is a period of time during which the software vendor plan to use a software component. We will fix the values of discounting interest rate, number of usage and maintenance cost. In this test, we will use "*jsockets*" as a software component that we need. All the value we chooses to test

Testing input:

Software component – jsockets

Discounting interest rate - 30%

Number of uses under one time unit period = 3 times

Maintenance cost – 20 virtual money units

We will input a list value of the *length of planning horizon*, these values with 0.5 apart in our time-unit :

1, 1.5, 2, 2.5, 3, 3.5, 4, 4.5, 5, 5.5

Testing output

The user agent made buy/rent decisions as following:

rent, rent, rent, rent, rent, rent, buy, buy, buy, buy.

Based on this testing result, we can draw follow Figures. Figure 4.1 is decision made against length of horizon, Figure 4.2 is total cost against length of horizon:

Figure 5.1 Buying/Renting Decision VS Length of Horizon

From the Figure 4.1, we can see that the user agent's buy/rent break-even point (break-even point – means the total costs at this point for renting and buying are equal.) is between 3.5 and 4 under previous fixed values for other three parameters. This test verified that the longer time the human user plans to use a software component, the more likely the user agent will decide to buy it, vice versa.

- Discounting interest rate

For the *discounting interest rate*, we will vary the values of discounting interest rate while fixing the values for the *length of the horizon*, the number of uses and the *maintenance cost*.

Testing input:

Software component - jsockets

Length of the planning horizon – 3 time units

Number of uses under one time unit period = 3 times

Maintenance cost – 20 virtual money units

Discounting interesting rate vary by 5% apart:

5%, 10%, 15%, 20%, 25%, 30%, 35%, 40%

Testing output

buy, buy, buy, buy, rent, rent, rent, rent.

We can see that the break-even point is between 20% and 25 % for discounting interest rate. Following diagram illustrates this result.

Figure 5.2 Buying/Renting Decision VS Discounting Interest Rate

From Figure 5.2, this test verified that the higher discounting interest rate, the more likely the user agent would decide to rent it, vice versa.

- Number of uses

We will test the sensitivity of *number of uses* within one time unit on the buying/renting decision-making problem of the user agent.

Testing input:

Software component – jsockets

Length of the planning horizon – 1 time units

Discounting interest rate - 20%

Maintenance cost – 20 virtual money units

Number of uses values input as:

1, 2, 3, 4, 5, 6, 7, 8, 9

Testing output:

rent, rent, rent, rent, buy, buy, buy, buy, buy, buy.

Following Figure 5.3 illustrates the decisions against the number of usage.

Figure 5.3 Buying/Renting Decision VS Number of Uses

From Figure 5.3, We can see that under current testing values, the break-even point is at 4, once the number of uses within one time unit over 4, then the user agent will decide to buy this software component – “jsockets” instead of renting it. This testing result verified that the more times the human user plans to use this software component, the more likely the user agent would decide to buy it.

- Maintenance cost

Maintenance cost is an expense that the human user will spend to maintain this software component under one time unit when the human user decide to buy this software component.

Testing input:

Software component – jsockets

Length of the planning horizon – 2 time units

Discounting interest rate - 20%

Number of uses under one time unit period = 3 times

Maintenance cost inputs:

5, 10, 15, 20, 25, 30, 35, 40

Testing output:

Buy, buy, buy, rent, rent, rent, rent, rent.

Following Figure 5.4 illustrates the decision of buy/rent with *maintenance cost*.

Figure 5.4 Buying/Renting Decision VS Maintenance Cost

From Figure 5.4, the break-even point is between 15 and 20 under the current fixed values of other three parameters. This test verified that the more expensive to maintain this software component, the more likely the user agent will decide to rent it, vice versa.

In conclusion, these four parameters: *Length of the planning horizon*, *Discounting interest rate*, *the number of uses* and *maintenance cost*, are four decisive factors for the user agent to make decision on whether to buy or rent a software component. The combination of these four parameters and current market buying/renting prices will make the user agent to decision whether to buy or rent the specific software component from the marketplaces.

Test Case 3. Selling Decision Test of User Agent

A software component can be built from several sub-components as we described in chapter 1. The selling decision model will be based on comparison of the total cost of sub-components and current selling price for the software component that the human user wants to sell. This test is to examine this selling decision functionality of the user agent.

Testing input:

Software component – “inetfactory” which is component we want to sell.

Sub-components – jsobjects, 2d/3d/graphb. These two components are used in building “inetfactory”.

Prefer price - 90

Acceptable price - 80

Testing output:

First, the user agent will display “Activate user agent...., send seller agent sell at price 90”. That means the user agent has made decision to sell this software component, which imply that the software vendor can make profit by selling this component at price they chooses.

After we lowered the our selling price to 60, the user agent will display *“According to current market condition, you will not make any profit by selling the inetfactory. Abort the Action”*.

This message means that the user agent has decided not to send any seller agent because the current cost is exceeding the selling price of the software component – “inetfactory”.

The test proves that the user agent’s selling decision function is working to protect the human user’s investment so that he or she won’t lose money by selling any software component.

This chapter described a few functional evaluation tests that prove our e-marketplace working as we expected in our system design descriptions. Next

chapter will draw a conclusion of this research and present some future possible research topics.

Chapter 6

Conclusion

In previous chapters, we have presented an intelligent mobile agent based e-marketplace for software components trading. In this chapter, a summary of the research and possible future research topics will be described.

6.1 Summary

In this thesis, we built an e-marketplace for trading software components by using a mobile agent technology. This e-marketplace is designed and deployed in a distributed architecture in which each marketplace independently trades a certain number of software components and traders (buyers and sellers) can buy/sell in any number of marketplaces. Each marketplace has ability to automatically match the buyers and sellers based on their trading prices. By integrating a total cost business decision model into this e-marketplace, this e-marketplace can also provide decision making support capabilities to the users on whether to buy or rent and when to sell a software component from/to the marketplaces. To achieve this decision support goal, this e-marketplace is implemented with an intelligent user agent

that can perform decision-making processes for the users. For software component buyers, the user agent makes buying/renting decision by incorporating four key parameters: current discounting interest rate, length of planned horizon, number of uses, and maintenance cost of the software component with current market prices by using a total cost calculation. For sellers, the user agent compares total cost of the sub-components with current market price of the software component to make decision on when to sell. Once the decisions on buying/renting or when to sell are made, a buyer agent or a seller agent will be sent to marketplaces to automatically trade the software component.

To the best of our knowledge, from e-marketplace's design point of view, the agent based marketplace most like our e-marketplace is InterMarket [14]. As we described in chapter 1, InterMarket integrates mobile agents and intelligent stationary agents to provide human users a trading decision-making capability. Within the InterMarket system, the mobile agents are used to pass trading instructions from the human user to the stationary agents that live in marketplaces. The stationary trade agents will make trade decisions based on the information provided by the mobile agents. The InterMarket system focused on two aspects: 1. The InterMarket can be accessed from portable mobile device (cell phone etc.) using mobile agents. 2. Intelligent stationary agent that lives in the marketplace makes intelligent human-like trade decisions.

While our e-marketplace is designed for software components trading, it has different goals from the InterMarket. Our e-marketplace has decision-making functionality integrated into the user agent instead of the trading agent, and its decision-making business model is based on the total cost calculation model, which is different from InterMarket's human-like decision model. In our e-marketplace, the trading agents (buyer agent and seller agent) are moving from one marketplace to another, whereas the trading agent in InterMarket always lives in one marketplace, only its mobile messaging agent moves from one market to another market.

On the other hand, from studying software buying and renting problem point of view, our approach in this e-marketplace project is different from [4], in which Marchand and Jacobsen developed an economic model that studied the effects of software selling and leasing actions of an independent software vendor on the profitability of an application service provider. Their approach is from independent software vendor's perspective. Their main result is that the introduction of competition (software leasing VS software licensing) lowers the prices of the software licenses as well as the price of the ASP services [4]. Our approach is from the software developers' point of view (software buyers). Our result is that the software component buyer's decision on whether to buy or rent a software component is based on four key factors: discounting interest rate, length of planned horizon, number of uses and

maintenance cost. The combination of these four factors will decide whether to buy or rent a software component from marketplaces.

5.2 Future Research

In this research, we only focused on the user agent's capability to make buying/renting decision. One of the future research topics could be to build a new intelligent function for the user agent, so that it can make future predictions and recommendations based on its analysis of current software component marketplaces.

In this e-marketplace, we only consider the price of software components as way to match buyer and seller. One of future research topic might be to build an evaluation system that will compile the software price, performance, security and user preference, so that software component buyer and seller can be matched based on their individual needs. In other words, software vendors can be more satisfied by finding exactly what they need in the market.

In current system, it is up to buyers to manually download purchased product from sellers, so another direction of the future research could be to build an automatically product transaction system that will enforce product transaction from sellers to buyers by using a secure channel.

Bibliography:

- [1] Howard Baetjer, Jr. *Software As Capital*, An Economic Perspective On Software Engineering, IEEE Computer Society Press.
- [2] Monahan, G., *Management Decision Making*, Cambridge, 2000. pp 10 – 26.
- [3] Jacobsen, H-A., *Middleware for software leasing over the Internet*, ACM Conference on Electronic Commerce, 1999
- [4] Marchand N., and Jacobsen, H.-A., *An Economic Model for Studying Dependencies between Independent Software Vendors and Application Service Providers*, *Electronic Commerce Research*, 1, 315-334, 2001, Kluwer
- [5] ADK2.1, Tryllian Technology Inc. Tryllian B.V., The Netherlands Amstel Business Park Joop Geesinkweg 701 www.tryllian.com
- [6] Wayne E. Etter and Fred F. Caldwell, *Buy or Lease? Commercial Property Decisions.*, the real estate center journal, spring 1995 publication 1064 instructor's notebook.
- [7] Cameron Ross Dunne, *Using Mobile Agents for Network Resource Discovery in Peer-to-Peer Networks* , School of Computer Applications, Dublin City University, Dublin 9, Ireland.
- [8] P. Maes, R. Guttman, and A. Moukas. *Agents that buy and sell: Transforming commerce as we know it*. *Communications of the ACM*, 42(3):81–91, Mar. 1999.
- [9] Anthony Chavez, Pattie Maes, *Kasbah: An Agent Marketplace for Buying and Selling Goods*, MIT Media Lab, Cambridge.

[10] Peter R. Wurman Michael P. Wellman William E. Walsh, *The Michigan Internet AuctionBot: A Configurable Auction Server for Human and Software Agents*, University of Michigan Artificial Intelligence Laboratory
<http://auction.eecs.umich.edu/>

[11] Deugo., D., Weiss, M., and Kendall, E., *Reusable Patterns for Agent Coordination* . In: Omicini, A., *Coordination of Internet Agents*, Springer, 2001

[12] Weiss, M., *Patterns for e-Commerce Agent Architectures: Using Agents as Delegates* , Pattern Languages of Programming (PloP), 2001

[13] Ryszard Kowalczyk , Mihaela Ulueru, Rainer Unland, *Integrating Mobile and Intelligent Agents in Advanced e-Commerce: A Survey*, Intershop Research, Intershop Tower. University of Calgary, Electrical Engineering, University of Essen Schützenbahn.

[14] Ryszard Kowalczyk, Bogdan Franczyk, Andreas Speck, *InterMarket – Towards Intelligent Mobile Agent e-Marketplaces* Intershop Research
Intershop Tower D-07740 Jena, Germany

[15] Dasgupta, P.; Narasimhan, N.; Moser, L.E.; Melliar-Smith, P.M., *MAGNET: mobile agents for networked electronic trading*, IEEE Transactions on Knowledge and Data Engineering, Volume: 11 Issue: 4, July-Aug. 1999, pp. 509–525. <http://alpha.ece.ucsb.edu/~pdg/magnet/>

[16] PR Wurman, WE Walsh, MP Wellman, *Flexible double auctions for electronic commerce: Theory and implementation*. Decision Support Systems 24:17-27,1998.

[17] N. Fukuta, T. Ito, T. Ozono, and T. Shintani. *A Framework for Cooperative Mobile Agents and Its Case-Study on BiddingBot*, In the Proceedings of the JSAI 2001 International Workshop on Agent-based Approaches in Economic and Social Complex Systems (AESCS 2001), pp.91—98.